TU Clausthal

Clausthal University of Technology

Model-Based Simulation Systems Engineering

Umut Durak

Umut Durak

Model-Based Simulation Systems Engineering

Cumulative Habilitation Thesis

approved by the
Faculty of Mathematics/Computer Science and Mechanical Engineering
Clausthal University of Technology, Germany
2018

Dedicated to my uncle, Mustafa Ozbek,
who never ceased encouraging me to be a scientist
ever since I was a small kid.

Preface

In 2007, I received my Ph.D. in Mechanical Engineering from Middle East Technical University (METU) in Ankara, Turkey. Until 2012, I worked as principal researcher and head of Modeling and Simulation Group in Defense Industries Research and Development Institute (SAGE) which is a part of Scientific and Technological Research Council of Turkey (TUBITAK). Then, for about one and a half years, I led the system simulations group of Roketsan Missiles Inc. In the second half of 2013, I joined the German Aerospace Center (DLR) Institute of Flight Systems. Since my Ph.D., I have been working on adapting model-driven methodologies for engineering of simulation systems. This will be my cumulative habilitation thesis, which embodies a summary of my work and a number of papers.

Clausthal-Zellerfeld, May 2016

Umut Durak

Contents

Part I

Summary

1

Introduction

1.1 Simulation Systems Engineering

The subject of modeling is the representation of a certain aspect of reality for a particular purpose through the creation of a model. Models may represent systems, entities, phenomena or processes [1]. Simulation then generates the system behavior from that model. Ören divides the motivations of modeling and simulation into three categories: a) experimentation, b) gaining experience, and c) entertainment [2, 3]. Simulation *experimentation* is conducted to achieve goals such as performance and behavior prediction, evaluation of alternatives, or sensitivity analysis. *Gaining experience* is another motivation for simulation that targets training for motor skills, decision and/or communication and operational skills under controlled conditions. The simulation experience is also used, for example in the gaming industry, for *entertainment*. Driven by these motivations, simulation is applied to various disciplines ranging from aerospace, agriculture and astronomy to transportation and waste treatment [4]. Furthermore, the contribution of simulation to a discipline x is known as *simulation-based x*. Simulation-based acquisition, engineering and education are some of the examples.

Synergies play a critical role in the evolution of disciplines. The evolution of simulation has also been significantly effected by the contributions from other disciplines [6]. The International Council of Systems Engineering (INCOSE) defines systems engineering as an interdisciplinary approach and a means to enable the realization of successful systems [7]. The contribution of systems engineering to simulation is by introducing methodologies that enhance the engineering of simulations, especially for large and complex systems. This contribution is called systems engineering-based simulation or *simulation systems engineering*, for short [8]. It is an interdisciplinary process for developing, maintaining and employing simulation systems [9].

The life cycle for a simulation study that was introduced by Balci in the early 1990s (Figure 1.1) was one of the early efforts towards developing simulation systems engineering process [5]. The process starts with the problem, for which a proposed solution technique is communicated. The life cycle then includes definition of the objectives and a modeling pipeline from conceptual model through communicative model, programmed model and experimental model. Finally, the simulation results lead to either redefinition of the objectives or a decision. Validation and verification activities through the life cycle are also introduced in Balci's study.

Subsequently, IEEE Std 1516.3-2003, IEEE Recommended Practice for High Level Architecture (HLA) Federation Development and Execution Process (FEDEP), proposed a process for federation development, particularly for distributed simulations that utilize HLA [11]. It met with a favorable reception as a starting framework for tailoring an end-to-end process for the

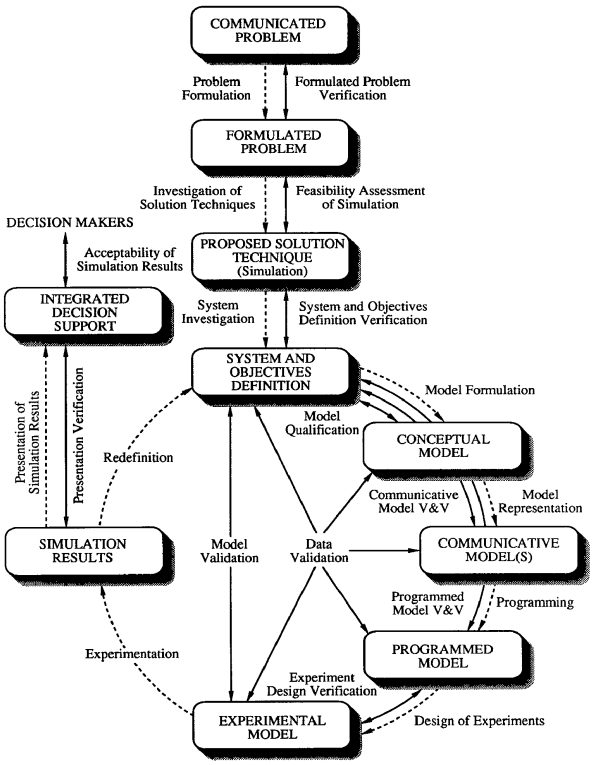


Fig. 1.1. The life cycle of a simulation study (© 1990 IEEE. Reprinted, with permission, from [5])

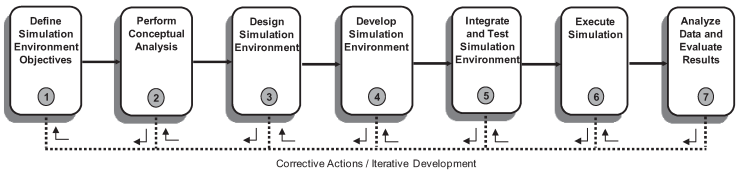


Fig. 1.2. Distributed Simulation Engineering and Execution Process (© 2010 IEEE. Reprinted, with permission, from [10])

development and execution of HLA federations. FEDEP was then generalized by the Simulation Interoperability Standards Organization (SISO) FEDEP Product Development Group (PDG) to support the engineering process for all types of distributed simulation development and execution efforts (Figure 1.2). In 2010, this was published as IEEE Std 1730-2010: IEEE Recommended Practice for Distributed Simulation Engineering and Execution Process (DSEEP) [10]. It is currently the state-of-the-art and most well-received standard process for engineering of simulation systems.

The first step of the DSEEP-recommended process is to define simulation environment objectives. The requirements engineering study is introduced as the second step. Scenario development, conceptual modeling, and definition of requirements are carried out in this step. The third step is designing the simulation, in which the members or the participants of the distributed simulation are selected. Reuse candidates are identified as well as the ones to be newly developed. Then,

the requirements are allocated to the member applications. The final part of this step is the design of member applications. In the fourth step, new member applications and modifications to the existing ones are implemented. Integration and testing of the simulation environment are carried out in the fifth step. In the sixth step, the distributed simulation is executed. The process concludes with the seventh step in which data analysis and evaluation are conducted.

1.2 Model-Driven Methodologies for Simulation Systems Engineering

Inspired by the basic principle of object technology (*"Everything is an object"*), Bézivin postulates *"Everything is a model"* as the basic principle in order to create a research agenda for the utilization of models in the engineering of software intensive systems [12]. Models can be descriptive or prescriptive. While descriptive models are used to specify the reality of systems, prescriptive models define how a system should be implemented [13]. We use the descriptive models of the systems to simulate them and prescriptive models of the systems to build them [14]. INCOSE defines Model-Based Systems Engineering (MBSE) as the formalized application of modeling to support system requirements, design, analysis, verification and validation throughout the life cycle of systems [15]. Aligned with the other contributions of systems engineering to simulation discipline, this thesis investigates further possibilities for leveraging the utilization of models in systems engineering, particularly as applied for software intensive systems, in engineering of simulation systems. The motivation is to improve productivity with the generation of simulation systems engineering artifacts, including simulation software code through transformation and stepwise refinement of models [16].

This thesis is based on the categorization that Brambilla and his colleagues introduced for the model-based and model-driven approaches in the engineering of software intensive systems [13]:

- Model-Driven Development (MDD) proposes a paradigm that utilizes models as the primary artifacts and redefines the implementation as (semi)automatic generation from the models. The process relies on the use of models, and the systematic production and transformation of models.
- Model-Driven Architecture (MDA) is the particular vision of MDD proposed by the Object Management Group (OMG).
- Model-Driven Engineering (MDE) expands MDD to cover all engineering process areas.
- Model-Based Engineering (MBE) utilizes model-driven practices pragmatically, not necessarily in an integrated fashion, in various steps of the engineering process. While models are important in MBE, they do not necessarily drive the development process. In this sense, all model-driven processes are regarded as model-based.

Model-driven methodology proposes the development of models and generation of executable software entities through successive model transformations [17]. The key ingredients of model-driven methodologies are modeling languages, metamodels and transformations [13]. Modeling languages enable the definition of a concrete representation for a model and metamodels are used to define modeling languages. Transformations are described as the mappings between models which are specified at metamodel level.

The Unified Modeling Language (UML) has evolved in the last two decades to be the standard specification for modeling software structures, behavior and architecture [18]. With UML and utilizing Meta Object Facility (MOF) [19], the key foundations for an integrated approach (known, in this case, as MDA) from business or domain models, through logical system models to

implementation models are provided [20]. Likewise, utilizing UML and Ecore, Eclipse Modeling Framework (EMF) has been developed as an alternative framework within the Eclipse ecosystem for MDD [21]. Furthermore, UML has been extended to OMG System Modeling Language (SysML) in order to support modeling for complex systems which further include hardware, information, personnel, procedures, and facilities [22]. Thus, applying model-driven approaches for systems engineering is empowered.

From Petri-nets [23] to bond graphs [24], the modeling and simulation community has a long lasting experience in modeling languages. Modeling, creating abstractions of systems and processes is their core business. Employing modeling and model-driven practices in engineering of simulation systems, therefore, suits this community perfectly.

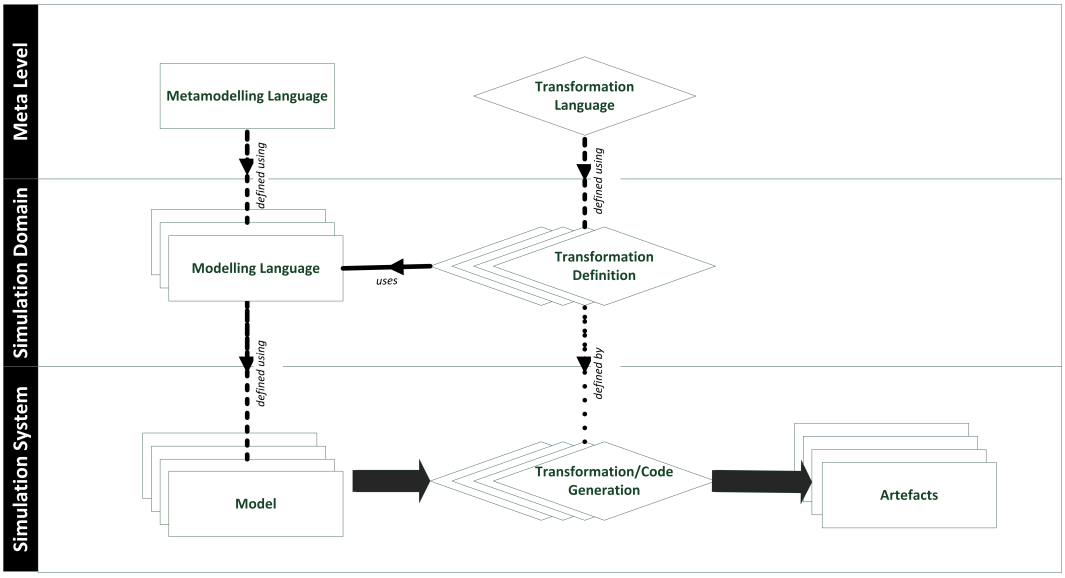


Fig. 1.3. Overview of model-driven methodology (Adapted, with permission, from [13])

Figure 1.3 is adapted from [13] in order to present an overview of a model-driven methodology for simulation systems. The conceptualization of models and transformations are provided in *meta level*. At the *simulation domain* level, the definition of modeling languages and the transformations are provided together. The models and the transformations are the concepts of the *simulation system* level where new artifacts, such as other models or simulation code, are generated.

1.3 Related Work

The idea of using model-driven methodologies in simulation dates back to 2002, when Tolk proposed merging the concepts and ideas of HLA into MDA [25]. This position paper claimed that MDA would influence the future of modeling and simulation. In 2003, Parr and Keith-Magee published a complementary article to [25] presenting the motivation of applying MDA in modeling and simulation, and discussing possible application approaches [26].

There have been various efforts since then, which I would like to categorize into two groups. The first group of efforts proposes model-driven approaches, methodologies, tools and techniques

to be employed in the particular steps of the engineering process of simulation systems in order to solve particular problems. While it is impossible to provide an exclusive review of these efforts, which span more than a decade, the important ones within the context of this thesis are summarized as follows.

In 2002, Lara and Vangheluwe presented their interactive tool, ATOM^3, by which they supported model-driven practices, metamodeling and explicit model transformations for simulation modeling [27]. With ATOM^3, they aimed to contribute to multi-paradigm modeling by enabling model transformations between different formalisms.

In 2004, Computer Automated Multi-Paradigm Modeling (CAMPaM) was introduced as a model-based development based on a combination of metamodeling and graph transformations [28]. Then ATOM^3 was employed for the CAMPaM of traffic networks in [29]. A domain specific formalism *traffic* is developed for vehicle traffic network modeling via metamodeling. The transformation to *Petri-net* models is then accomplished using the graph transformation capabilities of ATOM^3. While the overall simulation engineering process was not within the scope of CAMPaM, it addressed the employment of model-driven methodologies in multi-paradigm modeling by providing tools and presenting application use cases.

Following [25, 26], in 2006 Ozhan and Oguztuzun introduced their model-driven methodology for designing simulation data exchange model for HLA-based federations [30]. They proposed metamodeling for conceptual model and data exchange model, namely for the field artillery domain (FA) and HLA Object Models (HLA-OMT) and promoted model transformations from FA to HLA-OMT [31]. In 2008, Topcu et al. proposed the Federation Architecture Metamodel (FAMM), which also enables a behavioral description of federates based on life sequence charts [32]. They announced that the benefits of FAMM were to provide a basis for code generation and static analysis of federation architectures. In 2010, the same authors published their article which utilizes FAMM for code generation for HLA compliant federates [33].

Duarte and Lara presented their model-driven approach to agent based simulation in 2009 [34]. They proposed a domain-specific visual language for agent-based simulation in Eclipse and employed code generation practices in order to obtain code for an agent-based simulation environment, MASON.

In 2011, Mittal and Douglass proposed the utilization of model-to-model transformations for generating Discrete Event System Specification (DEVS) Modeling Language from domain specific languages [35] and Ighoroje et al. introduced their DEVS-Driven Modeling Language promoting automatic code generation for simulation and formal analysis in 2012 [36].

Recently, there have been some efforts that address, not the development steps of simulation engineering, but the maintenance and evolution steps. Denil et al. augmented a popular model-based design tool, MATLAB/Simulink, with rule-based model transformation capabilities in 2014 [37]. They based their approach on a metamodel that they proposed for Simulink causal block diagrams. They then integrated an external model transformation library that utilizes this metamodel to execute transformation rules. Thereby, they equip the modelers to specify in-place transformation to automate their model refactoring tasks. Ledet et al. presented their model-driven approach for reverse engineering platform specific MATLAB/Simulink models to platform independent SysML models for model replicability and reproducibility [38]. Further, in 2008, Mittal and Zeigler addressed verification and validation by adapting a Model-Based Testing (MBT) approach for simulation [39]. They proposed automated test model generation from DEVS models.

The second group of efforts proposes an overall MDD process and promotes successive transformations from the early stages of simulation development, such as conceptual modeling or systems modeling through simulation design to the executable simulation code. A short review

of a selection from these efforts aligned with the context of the thesis is given in the following paragraphs.

In 2004, Iba et al. proposed an MDD process for agent-based simulations [40]. They promoted a three-step process that consists of conceptual modeling, simulation design and verification. They adapted various UML diagrams for these steps and propose a set of tools for these diagrams. While they did not address metamodeling and model transformation aspects, they endorsed a simulation development process with successive manual model refinement, enhancement and transformation through the abstraction levels towards the executable simulation.

In 2006, Guiffard et al. presented their CAPSULE project, in which they developed an MDA-based simulation development methodology for HLA compliant simulations [41]. They proposed metamodels for platform independent modeling and HLA specific modeling. Then, they benefited from model transformations and code generation. UML Profiles are utilized for metamodeling and model transformations are enabled using a tool that allows the construction of transformation rules following an XML grammar, in this case XML Metadata Interchange (XMI), in rule sets. Code generation was finally achieved using IBM Rational Rose capabilities.

D'Ambrogio and his colleagues presented their MDD approach to develop DEVS simulations in 2010 [42]. Their paper presents one of the earliest attempts that targets a complete model-driven simulation development pipeline from systems modeling to DEVS/JAVA code through successive (manual and automated) model-to-model and model-to-text transformations. UML and UML profiles are extensively utilized throughout this pipeline. While the paper presents a methodology for DEVS/SOA platform, which is an implementation of DEVS formalism over a Service Oriented Architecture (SOA), it also claims that the approach is general enough to be adapted to additional DEVS-based implementation technologies.

Cetinkaya, Verbraeck and Sect in 2011 proposed an MDD framework for modeling and simulation (MDD4MS) [43]. They presented a modeling and simulation life cycle which has five stages: (1) Problem Definition, (2) Conceptual Modeling, (3) Specification, (4) Implementation and (5) Experimentation. They additionally intended to provide an MDD framework to obtain continuity throughout this life cycle. This framework is composed of various metamodels and model transformations that range from the conceptual model to simulation code. An Eclipse Modeling Framework-based prototype is also provided. The framework supports Business Process Model and Notation (BPMN) for conceptual modeling, DEVS for specification, Distributed Simulation Object Model (DSOL) for implementation infrastructure and Java as the programming language. It consists of BPMN and DEVS metamodels and model-to-model transformations like BPMN to DEVS that are written in ATLAS Transformation Language. A specific model-to-text transformations for JAVA is also supplied with the framework.

1.4 Contribution

There are a wide range of methodologies and approaches used in simulation modeling. A comprehensive taxonomy for simulation modeling methodologies and approaches has been introduced in the discrete-event modeling and simulation ontology, DeMO that is developed by Silver et al. [44]. These methodologies and approaches basically differ depending on the behavior of the modeled system (e.g. continuous, discrete, hybrid), the focus of the modeler (e.g. activity diagrams, state transition diagrams), the abstraction (e.g. agent-based simulation, object-oriented simulation), the execution (e.g. activity scanning, event scanning) or model syntax (e.g. declarative, functional). While UML accommodates an adequate set of well-accepted formalisms for software and systems modeling (with SysML extensions), the diversity in methodologies and approaches in specifying simulation modeling assets, prohibits UML from being sufficient for the engineering

of simulation systems. Moreover, due to this issue of methodology and approach diversity, it has not been possible to develop a standard set of modeling languages for simulation. Therefore, although there have been far reaching efforts, as presented in the second group in the previous section, it is practically hard and maybe even not attainable to develop a common and widely accepted MDD process for engineering of simulation systems which achieves coverage from domain modeling through system modeling, to development.

This thesis exploits MBE and promotes pragmatic utilization of model-driven practices in the engineering of simulation systems, namely Model-Based Simulation Systems Engineering. It endorses model-driven practices as indispensable elements of the emerging simulation systems engineering tool set and asserts that pragmatic employment of these practices is beneficial in increasing the quality characteristics, such as effectiveness, efficiency, consistency and repeatability of the simulation systems engineering process. It supports this propositional statement with various publications that introduce model-driven techniques and methodologies which address particular steps in the simulation engineering process.

In 2009, Ozdikis, Oguztuzun and I presented a transformation approach for supporting model-driven development of simulations, in which the Computation Independent Model is represented as Web Ontology Language ontology [45]. As with the Platform Independent Model, UML models are automatically generated by executing rules that govern the mapping of OWL constructs to UML constructs. This paper presents, chronologically, the initial effort that later led to this thesis. In 2010, with the same colleagues, we presented our model transformation tool this time from OWL to HLA Object Models [46].

In 2014, I presented a paper with Topcu, Siegfried and Oguztuzun that addresses the scenario development. We proposed a metamodel for conceptual scenarios and exploited the utilization of model transformations from conceptual scenario to executable scenario and simulation environment design artifacts [47].

I published two other papers with my colleagues in 2014, which address integration of the simulation environment. The first one focused on utilization of Functional Mock-up Interface to generate HLA-complaint federates [48]. In the second one, Gerlach, Gotschlich and I presented our code generation process for MATLAB/Simulink models that is tailored from MATLAB/Simulink Generic Real-time Target in order to enable seamless integration in a target flight simulator environment [49].

I also started working on applying model-driven methodologies for testing simulation environments in 2014. In the first technical note that I published with Schmidt and Pawletta, the underlying metamodeling effort of model-based testing approach is presented [50]. In 2015, based on this metamodel and employing System Entity Structure / Model Base framework, we developed a model-based testing methodology [51]. And, finally, we applied this to objective fidelity evaluation of flight simulators [52].

I published two articles that investigate model-based methodologies for simulation evolution and modernization. In the first one that was published in 2015, I extended Knowledge Discovery Metamodel to support model-based reverse engineering of legacy simulation assets [53]. The second one that was published in 2016 presents a model transformations approach in Scilab to refactor Scilab/Xcos models [54].

Recently, Ozturk, Katircioglu and I attempted to extend the utilization of models in engineering of simulation systems and proposed the use of causal block diagrams in MATLAB/Simulink for a model-based deployment process [55].

In 2016, Topcu, Oguztuzun, Yilmaz and I published a book on distributed simulation that presents a model-driven engineering approach which refers to almost all the studies mentioned above [14].

1.5 Structure of the Thesis

In order to create a common understanding of the concepts and processes, the thesis first provides an ontology for simulation systems engineering based on DSEEP in Chapter 2. Then, in Chapter 3, it introduces Model-Based Simulation Systems Engineering by presenting particular studies that exploit one or more applications of model-driven practices in most of the steps of the simulation systems engineering process. Finally, a discussion of outlook and future directions is provided in Chapter 4.

2

Ontology of Simulation Systems Engineering

It is necessary to have a shared understanding of simulation systems engineering concepts and process before introducing Model-Based Simulation Systems Engineering, which is essentially model-driven methodologies applied to particular steps of this process. Ontologies provide us with explicit specifications for our conceptualizations [56]. Their merits include systematization of knowledge and the creation of a common vocabulary [57]. This chapter is based on our paper:

> Umut Durak and Tuncer Oren. 2016. Towards an ontology for simulation systems engineering. In *Proceedings of the 48th Annual Simulation Symposium*, Pasadena, CA, 163-170.

In this paper, we first highlight the synergies among the disciplines as the important players in the their evolution. Accordingly, modeling and simulation discipline has been evolving pertaining to its synergies with computerization and software engineering, artificial intelligence and software agents, system theories, soft computing, and systems engineering. Simulation systems engineering is an outcome of the synergy between systems engineering and simulation.

DSEEP is a tremendous contribution towards the standardization of the simulation systems engineering life cycle process. It specifies the seven-step process with their corresponding activities. Then, it introduces activity descriptions with their inputs, outputs and a list of recommended tasks. Furthermore, it defines the roles that take a part in the steps of the process. Data flow diagrams are employed by DSEEP in order to graphically represent the product flow among the activities, and to introduce the data stores.

Although data flow diagrams help a lot in presenting the information in a structured way, DSEEP is still far from providing an unambiguous, systematic and structured explanation of the shared terms and vocabulary for coordination, cooperation and integration of simulation systems engineering processes. Following the promise of ontologies in creating shared conceptualization, we attempted to develop an ontology for simulation systems engineering based on DSEEP.

In virtue of their popularity, Protégé [58] is used as the ontology development environment and ontologies are developed using the Web Ontology Language (OWL) [59]. As depicted in Figure 2.1, at the top level they are Activity, Data Store, Information, Product, Role, Step and Task. Each entity is then further inherited to its child entities. All the concepts that are introduced in DSEEP are captured in the ontology.

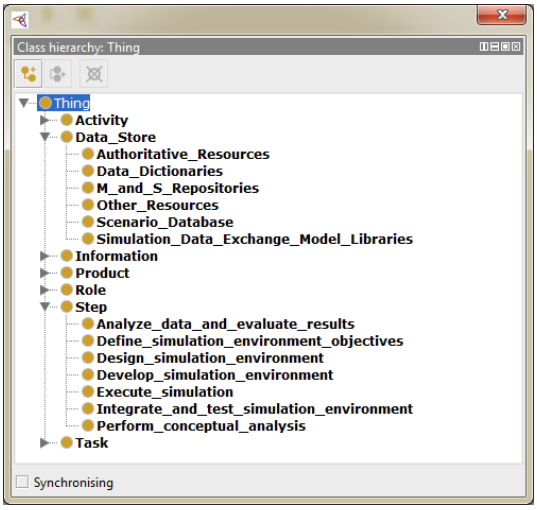


Fig. 2.1. The hierarchy of the simulation systems engineering ontology [9]

While machine processability is envisioned as the enabler for the future ontology-based simulation process integration research, human readability is regarded as the quality that will guarantee the creation of common and shared conceptualizations. So the understandability of the constructed ontology by human readers is regarded as important as its machine processability. Therefore, Visual notation for OWL ontologies (VOWL) [60] which is a visual language for the user-oriented representation of OWL ontologies, is investigated as an ontology presentation medium. While it possesses flaws of early development, basically in filtering, query and navigation, VOWL is regarded as promising for ontology visualization.

This study contributes with attempting to create a shared conceptualization about simulation systems engineering process using an ontology. It envisions enabling interoperability and cooperation within the simulation systems engineering process with machine-readable shared vocabulary. The reader can find the paper in Appendix B.1.

While it promotes DSEEP as the baseline, it also highlights the points where further discussion and elaboration is required to come up with a well-accepted taxonomy. Aligned with that, while DSEEP provides a valuable reference for a simulation systems engineering life cycle, it still fails to provide full coverage of all process areas. Therefore, I would like to refer here to *ISO/IEC 12207 Systems and Software Engineering - Software Life Cycle* [61], which proposes a more comprehensive life cycle, but for software systems. There is currently no literature that presents a comprehensive comparison of DSEEP and ISO/IEC 12207. However, if we only look at *Technical Processes*, claiming that the process areas related to installation, maintenance and disposal are not addressed in DSEEP, we will not be wrong.

3

Model-Based Simulation Systems Engineering

This chapter renders 10 papers that present model-driven practices applied to particular steps of simulation systems engineering. These papers adduce the benefits of applying model-driven practices in increasing the efficiency, effectiveness, consistency and repeatability of simulation systems engineering process. One paper is presented for each of the *conceptual analysis*, *simulation environment design*, and *simulation environment development* steps of DSEEP. For *integration and test of simulation environment*, I will be presenting four papers. Regarding the systems engineering processes areas that are not covered by DSEEP, this chapter includes two particular papers for applying model-based methodologies in *simulation maintenance* and one for *simulation installation*.

3.1 Conceptual Analysis

Conceptual analysis is performed immediately after defining the simulation environment objectives. DSEEP defines the aim of this step as developing a representation of the real-world domain and developing a scenario. It is composed of three activities: developing scenario, developing conceptual model and developing simulation environment requirements.

The purpose of scenario development activity is to construct a functional specification of a scenario. In 2012, Siegfried et al. developed the concepts operational scenario, conceptual scenario and executable scenario [62]. They defined operational scenario as a rough description of the intended situation and its dynamics, which are provided in the early stages of a simulation environment development process by the user or the sponsor. Such a scenario shall be refined and augmented with additional information pertaining to simulation. Conceptual scenario is then defined as a detailed specification of a scenario conforming to a formal metamodel. It shall be complete and consistent. Finally, the executable scenarios are the machine readable specifications which are used in simulation execution. In 2013, I was also involved in this study and we further investigated utilizing Base Object Model (BOM) for specifying conceptual models [63]. Later, I found the workflow from the operational scenario to executable scenario suitable for applying model-driven practices and proposed a follow up study in which we could construct a BOM-based metamodel for conceptual scenarios and exercise model transformations for executable scenarios and further possible targets. This section will introduce this effort based on the following paper:

Umut Durak, Okan Topcu, Robert Siegfried and Halit Oguztuzun. 2014. Scenario development: A model-driven engineering perspective. In *Proceedings of the 4th International Conference on Simulation and Modeling Methodologies, Technologies and Applications (SIMULTECH)*, SCITEPRESS, Vienna, Austria, 75-82.

We structured the model-driven scenario development process upon the four-layer metamodeling architecture of OMG, which specifies four levels: Information (M0), Model (M1), Metamodel (M2) and Meta-metamodel (M3), and their relations [19]. The process advocates constructing a Conceptual Scenario Metamodel prior to developing conceptual scenarios. The aim is to start with a metamodel to enable modeling a conceptual scenario, and then to support the model transformations from the source, conceptual scenario to target simulation application design, simulation environment agreements, and executable scenario.

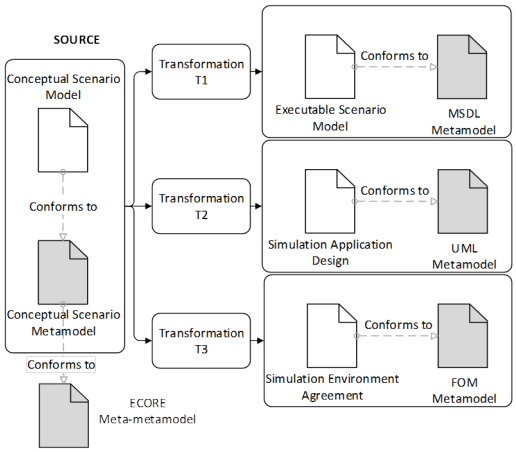


Fig. 3.1. Model-driven scenario development [47]

We exemplified the proposed process with Figure 3.1. We introduce a BOM-based metamodel for conceptual scenarios; Federation Object Model (FOM) metamodel [32] for simulation environment agreement, UML metamodel [18] for simulation application design, Military Scenario Definition Language (MSDL) [64] for executable scenario. Then, the conceptual scenario is subject to either model-to-model or model-to-text transformations which are specified with the mappings between the constructs of the source metamodel and those of the target metamodel.

In the sample implementation, Eclipse Modeling Framework (EMF) [21] which is a framework for describing models and then generating other constructs, such as other models, code or text from them, is employed to realize the four-layer metamodeling architecture. EMF provides Ecore as the meta-metamodel for describing metamodels. Four Ecore classes, namely, EClass, EAttribute, EReference and EDataType are utilized. EClass is the modeled class with attributes and references. EAttribute is the modeled attribute with a name and a type. EReference specifies the associations between classes. EDataType is the type of an attribute. Please refer to the example that is presented in Figure 3.2. *ConceptualScenario* is an EClass that has associations which are defined by EReference constructs: *entities*, *stateMachines*, *interplays*, *events* and *identification*. They relate *ConceptualScenario* to other EClasses *Conceptual Entity*, *StateMachine*, *PatternOf Interplay*, *Event* and *ScenarioIdentification*, respectively. Based on a sample opera-

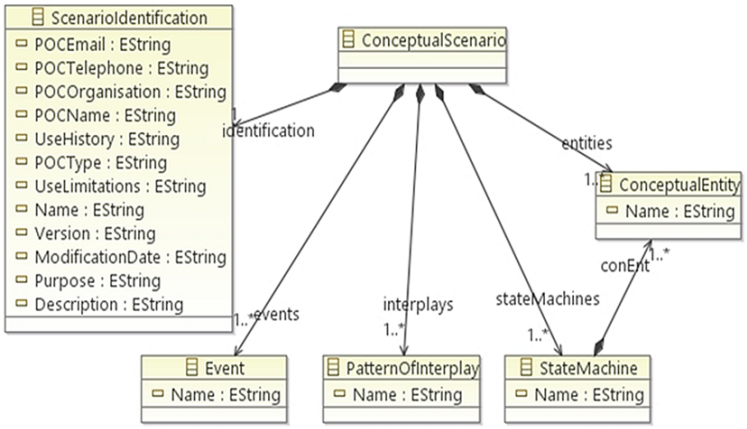


Fig. 3.2. Conceptual scenario metamodel top level diagram [47]

tional scenario from flight simulation domain, a conceptual scenario is then modeled using the constructed metamodel.

To define transformations from a source metamodel to a target metamodel, among the model transformation languages like Atlas Transformation Language (ATL) [65], Graph Rewriting and Transformation (GReAT) [66] and Query/View/Transformation (QVT) [67], QVT is selected for the sample implementation. Mappings are specified between object from the source metamodel for conceptual scenario to objects in the target metamodels for MSDL, UML or FOM.

The study provided valuable evidence for the applicability of model-driven methodologies in scenario development. A follow up effort, which aims to develop an aviation scenario definition language to support the application of model-driven scenario development in flight simulators, is in progress. The reader can find the paper in Appendix B.2.

3.2 Simulation Environment Design

DSEEP recommends a simulation environment design step which involves selecting member applications, designing the simulation environment and designing the member applications. In member application design, simulation systems engineers transform the knowledge that is captured in scenarios and conceptual models into a simulation application design. Referring to its similarity to software design, this step is regarded as a strong candidate for applying model-driven methodologies. Application of model transformations from conceptual scenario models to simulation application design in UML has already been presented in a previous section within the context of scenario development. This section introduces model transformations from a simulation conceptual model that is constructed as an ontology to simulation design and which is represented in UML base on the the following paper:

Ozer Ozdikis, Umut Durak and Halit Oguztuzun. 2009. User-guided transformations for ontology based simulation design. In *Proceedings of the 2009 Summer Computer Simulation Conference*, SCS, Istanbul, Turkey, 75-82.

In this study, the simulation conceptual model is regarded as a domain model and framework architecture is regarded as simulation design. Ontologies are a means to represent knowledge that is gathered during domain analysis for ease of both human understanding and machine processability. We proposed a tool support for the transformations across technical spaces, from the ontology which is the subject matter expert's view of the domain, to UML, which is the software developer's view of the domain.

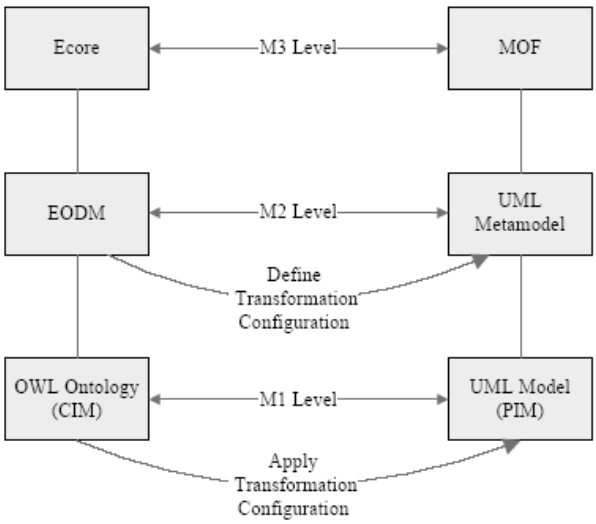


Fig. 3.3. OWL to UML transformation approach [45]

Based on MDA terminology, we can promote the ontology in OWL as Computation Independent Model (CIM), and the framework architecture in UML as the Platform Independent Model (PIM). As presented in Figure 3.3, the model transformation approach is located in reference to the four-layer metamodeling hierarchy of OMG MOF. MOF provides a meta-metamodel at the top level, M3. We utilized the standard UML metamodel in M2 layer which conforms to MOF. UML models conforming to the UML metamodel are used at layer M1. Finally, the M0 layer includes the instances created from a UML model. Similarly, the ontology modeling hierarchy is designed based on EMF. Ecore is used as the meta-metamodel at M3 layer. We then adopted the Integrated Ontology Development Toolkit (IODT) that is implemented by IBM for ontology driven development [68]. IODT provides EMF Ontology Definition Metamodel (EODM) which is an implementation of the OMG's Ontology Definition Metamodel. The transformation from CIM to PIM is defined from the ontology model as our source to the UML model as our target. With the OWL-to-UML transformation tool, we provided the capability to define the mapping rules between the EODM and UML metamodels at M2 layer. These rules are applied to a given OWL ontology to generate a UML class diagram.

The transformation is composed of transformation rules, mappings and constraints. A rule is defined as a collection of mappings from a specific OWL construct to some UML constructs, where a mapping is the prescription of how the target UML construct should be built using the source OWL construct. And finally, constraints restrict the entities to be evaluated in a specific rule or mapping.

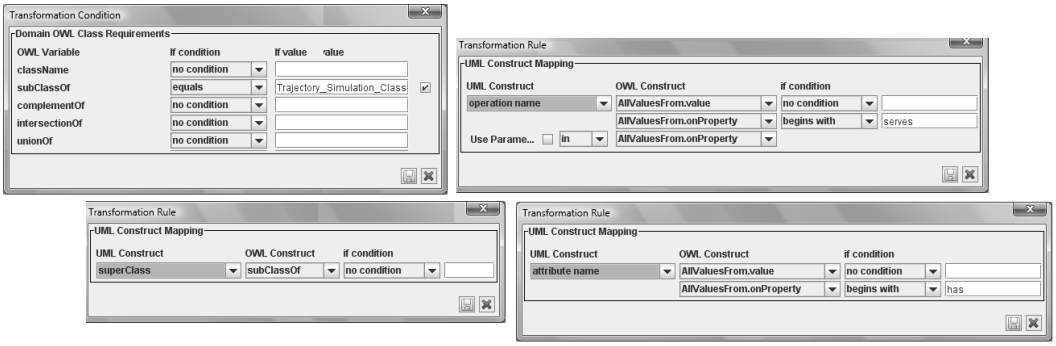


Fig. 3.4. OWL-to-UML tool screen shots from the case study [45]

The proposed methodology and developed tool is exercised with a case study in which Trajectory Simulation ONTology (TSONT) [69] which is a domain model for trajectory simulation, is transformed to design model. We defined two rules to fulfill the transformation requirements for the trajectory simulation ontology. In the first rule, we define mappings for UML classes and UML associations. The second rule is used for mappings to UML operation parameters, essentially, for setting the parameter names in the target. A set of screen shots from the transformation tool is presented in Figure 3.4.

We started constructing an ontology, namely TSONT, for the flight simulation of guided and unguided weapons, such as projectiles, rockets and missiles, as a part of my PhD thesis in 2005 [70, 69]. We used TSONT as a domain model in my PhD and constructed a simulation reuse strategy on top of it [71, 72]. After my PhD, extending the ideas in it, we started to use TSONT as a simulation conceptual model and investigated model transformations from a conceptual model, TSONT in this case, to particular assets of simulation systems engineering.

With this study we could claim that it is possible to generate member application design, typically in UML, from a simulation conceptual model using user guided transformations. While the generated member application design fails to provide detailed design features or platform specific details, this pragmatic application of model-driven methodology equips a simulation systems engineer with a baseline member application design that is consistent with his/her conceptual model. He/she can further elaborate it manually or with further model transformations. The reader can find the paper in Appendix B.3.

3.3 Simulation Environment Development

Simulation environment development is the next step after simulation environment design in DSEEP. It consists of developing simulation data exchange model (SDEM), establishing simulation environment agreements, and implementing member application designs and simulation environment infrastructure. Simulation conceptual models are not only used for member application design. They are also regarded as the source of information that is required to develop SDEM which specify the interactions of member applications. As an extension of the study presented in the previous section, this section introduces model transformations from a simulation conceptual model that is constructed as an ontology to SDEM that is represented as an HLA Object Model based on the following paper:

Ozer Ozdikis, Umut Durak and Halit Oguztuzun. 2010. Tool support for transformation from an OWL ontology to an HLA Object Model. In *Proceedings of the 3rd International ICST Conference on Simulation Tools and Techniques*, ICST, Malaga, Spain.

Object Models are regarded as HLA specific interface models which answer two important aspects of data communication policy; what to exchange, and how to exchange it [14]. In HLA, federates communicate via exchanging objects and interactions using a middleware, namely Run-time Infrastructure (RTI) which employs a publish/subscribe pattern. Publishing means declaring willingness to provide data, where subscribing means declaring interest in receiving certain data. RTI dynamically routes the data among publishers (producers) and subscribers (consumers). A Federation Object Model (FOM) describes the format and the structure of data and events that can be exchanged among federates in a federation execution in form of object and interaction classes.

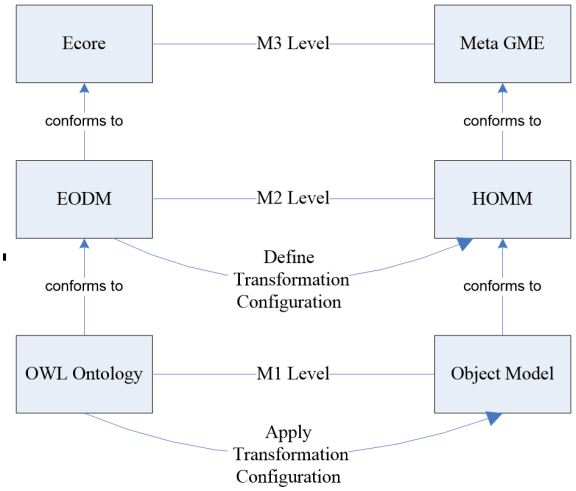


Fig. 3.5. OWL to FOM model transformation approach [46]

In this study, we promote a user guided transformation from conceptual model which is represented by an OWL ontology, to FOM. Presenting the model transformation approach in reference to the four-layer metamodeling hierarchy of OMG MOF, in the source, for the M2 layer we propose a scaled down version of HLA Object MetaModel (HOMM) which was introduced by Topcu et al. in [32] as a part of the Federation Architecture Metamodel (FAMM). FAMM employs metaGME which is the meta-metamodel provided in Generic Modeling Environment (GME) [73], in the M3 level. Object Models conforming to the HOMM, thus, to the HLA Object Model Template (OMT) are at the layer M1. Finally, the M0 layer includes the objects and interactions created during federation execution as instantiations of the Object Model. In the target, pretty much aligned with the paper that is presented in the previous section, Ecore is used as the meta-metamodel at M3 layer. EODM which is an implementation of the OMGs Ontology Definition Metamodel is employed at the M2 layer, where OWL ontology is placed at the M1 layer. As presented in Figure 3.5, our approach facilitates the definition of the mappings between

the EODM and HOMM at the M2 layer. These mappings are applied to a given OWL ontology to generate an HLA Object Model.

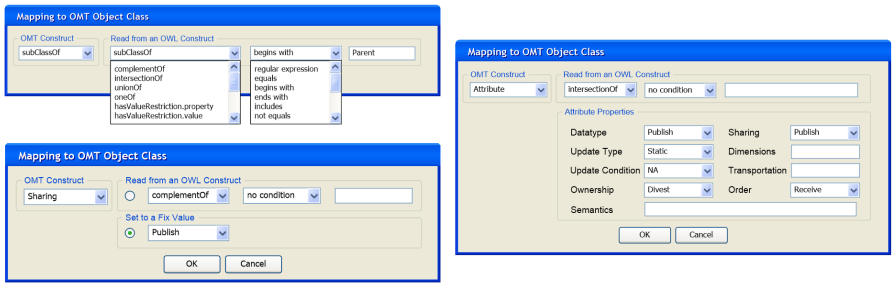


Fig. 3.6. OWL-to-FOM tool screen shots [46]

A tool is developed to enable the user to configure transformations. The transformation configurations are composed of mapping groups, mappings and constraints. A mapping group can be defined as a collection of mappings from a specific OWL (source) construct to some OMT (target) constructs. Source constructs can be OWL classes or OWL properties. A mapping is the prescription of how the target OMT construct should be built using the source OWL construct. The user can define four types of targets: Object Class, Attribute, Interaction Class and Parameter. Condition-value pairs can be applied on an OWL construct as constraints to restrict the entities to be evaluated in a specific mapping. Sample screen shots from the developed tool are depicted in Figure 3.6.

This study provides us with the opportunity to claim that model transformations can be utilized successfully to generate SDEM, in our case HLA Object Model from the simulation conceptual model. This not only advances productivity, but also helps considerably in building up the consistency of SDEM to the simulation conceptual model. As a follow-up to this study, Oguztuzun and I supervised a Masters thesis in which it was exercised in an end-to-end HLA federate development effort [74]. The reader can find the paper in Appendix B.4.

3.4 Simulation Environment Integration and Test

DSEEP recommends an integration and test step in which the members of the simulation environment are integrated and tested according to the requirements of simulation execution. In this section I will be presenting model-driven practices applied to integration and testing of member applications.

3.4.1 Integration

Integration refers to composing member applications into a simulation environment. It is enabled by the interoperability and composability of the member applications. Interoperability requires a means of communication between the interoperating entities; on the other hand, composability requires the specification of the interfaces to be composed [14]. They are technical challenges that need to be addressed during development. In this section, I will be presenting two studies that address these integration challenges.

Model Integration Workflow for Keeping Models up to Date in a Research Simulator

Code generation, or in other words, model-to-text transformation is a well applied model-driven practice that is more or less related to implementation phase where the member application code is generated for the particular model. Accordingly Mathworks is providing this capability for the simulation models that are constructed with MATLAB/Simulink [75]. In this study, we tailored the code generation process of MATLAB/Simulink and supported it with some automation scripts, and thereby addressed integration challenges by utilizing model-driven practices. The developed model integration workflow is presented based on the following paper:

> Torsten Gerlach, Umut Durak and Jurgen Gotschlich. 2014. Model integration workflow for keeping models up to date in a research simulator. In *Proceedings of the 4th International Conference on Simulation and Modeling Methodologies, Technologies and Applications (SIMULTECH)*, SCITEPRESS, Vienna, Austria, 125-132.

This study has its motivation from the flight dynamics model integration process in research flight simulators of the German Aerospace Center (DLR). The organization of a flight simulator is structured around the flight dynamics model. The other important components like control loading, instructor station, motion system, visual system, instrument displays either provide inputs to flight dynamic models or present their results to the pilot as cues [76]. The integration of multiple hardware and software components in real-time flight simulation is facilitated in DLR research simulators using the indigenous real-time distributed simulation framework, 2Simulate [77].

Although the code generation facilities provided by the modeling environments usually support a large set of target platforms, such as operating systems, compilers, hardware components like Input/Output devices, often the particular target of interest is not in that set. Either the target can be so special for the domain, like there is no predefined HLA target, which is particularly applied in distributed simulation, specifically in military simulation domain, or it can be application and organization specific, like in our case, in-house developed real-time simulation framework. For such cases, code generation facilities, like Simulink Coder [75] provides capabilities to define new code generation targets. In this study, we extended the Generic Real-time Target (GRT) from Simulink Coder to generate code for 2Simulate.

2Simulate is composed of 2Simulate Real-Time Framework (2SimRT), 2Simulate Model Control (2SimMC), and 2Simulate Control Center (2SimCC). While 2SimCC is the graphical user interface for simulation execution control, 2SimRT provides a software framework and an application programming interface for deterministic scheduling and controlling of real-time tasks. After code generation, Simulink models are also integrated to simulation environment as real-time tasks. The enabler of model integration is 2SimMC, which is is composed of 2Simulate Model Control Source (2SimMC-Source) and 2Simulate Model Control Scripts (2SimMC-Scripts). 2SimMC-Source abstracts model interfaces for 2SimRT and 2SimMC-Scripts include Simulink Coder Target Language Compiler (TLC) scripts to specify the 2Simulate target and MATLAB scripts to conduct the code generation and build process.

TLC script can be introduced as an interpreted programming language that converts a model description into code [78]. It applies a template-based model-to-text transformation approach where metamarkers/tokens are used to query the dynamic content from the model. Figure 3.7

```
#ifndef _%<Name>TSimSimulinkModelDefines_H_
#define _%<Name>TSimSimulinkModelDefines_H_

#define MODEL %<CompiledModel.Name>
#define RT
#define NUMST %<CompiledModel.NumSynchronousSampleTimes>

#define NCSTATES %<CompiledModel.NumContStates>
```

Fig. 3.7. An excerpt from a TLC script [49]

provides a sample that shows how TLC tokens are used to get information from the model and incorporate them in the source code.

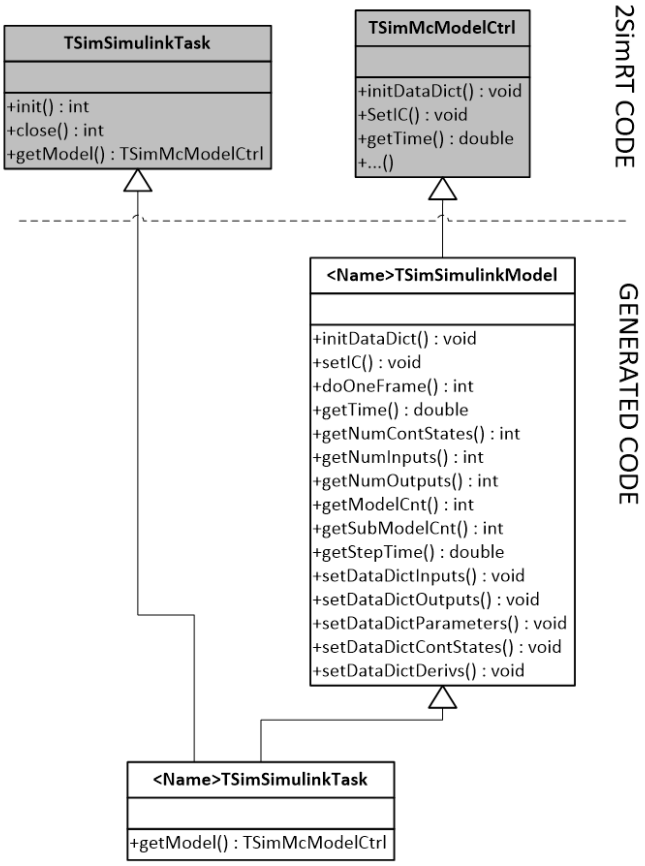


Fig. 3.8. The relation of generated code and 2SimRT [49]

In this study, we developed the TLC scripts that constitute 2SimMC-Scripts in order to extend the 2SimRT API and glue it with code that is generated using GRT. Thus, the generated code can be scheduled and executed as a real-time task by the simulation framework, 2Simulate. A class diagram that represents the relation between the generated code and 2SimRT is given in

Figure 3.8. The proposed methodology and developed infrastructure is currently been actively used in DLR.

With this study, we can attest that model-to-text transformations are useful in addressing the technical challenges of integrating member applications. With tailoring and augmenting the code generation process in the implementation phase, it is achievable to aspire integration-ready member application and thus, conclude an effective integration process. The reader can find the paper in Appendix B.5.

Adapting Functional Mock-up Units for HLA-compliant Distributed Simulation

There are some recent efforts for the standardization of model interfaces to tackle the integration challenges. Functional Mock-up Interface (FMI) [79] and Simulation Model Portability 2 (SMP2) [80] are two important ones. In this study, we focused on FMI. While various simulation modeling tools are providing code generation capabilities that target FMI, the integration of member applications that supports FMI, namely Functional Mock-up Units (FMUs), to HLA federations was not addressed. In this study, we adapted the layered simulation architecture of Topcu et al. [81] and proposed a configurable communication layer that enables integration of FMUs as FMU federates in HLA federations. This section is based on the paper:

> Faruk Yilmaz, Umut Durak, Koray Taylan and Halit Oguztuzun. 2014. Adapting Functional Mockup Units for HLA-compliant distributed simulation. In *Proceedings of the 10th International Modelica Conference*, Lund, Sweeden, 247-257.

FMI provides two standard interfaces, namely, FMI for Co-Simulation and FMI for Model Exchange [82]. While FMI for Model Exchange specifies the interface for callers with explicit or implicit integrators, FMI for Co-Simulation specifies the interface for simulation runnables that possess solvers in them. Regarding that HLA Federates as standalone simulation runnables, in this effort, we investigated FMI-Co-Simulation interface for federate development.

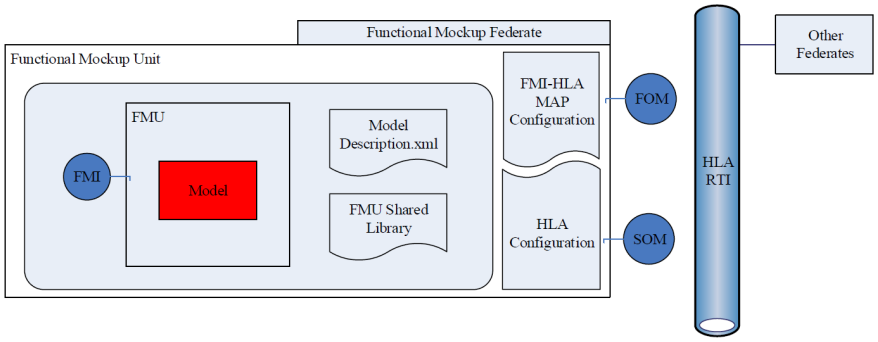


Fig. 3.9. Functional Mockup Federate [48]

The challenge raises due to fundamental differences between the world views of two standards. While FMI provides a standard access for the model execution services of a basic hybrid model,

HLA focuses on enabling the integration of models via specifying data communication policies. The gap between them motivated us to develop an adaptation layer for configuring and mapping the concepts. We aimed at supporting FMU code generation with a technique that will deliver it to distributed simulation purposes as a federate. An overall description of FMU federate is given in Figure 3.9.

The adaptation layer can be regarded as a wrapper that conducts the following functions:

- Instantiating, initializing, stepping and terminating an FMU model
- Communicating with the other member applications utilizing the HLA standard interface
- Converting FMU model outputs to compatible HLA data types and sending them as HLA object updates
- Receiving model inputs from HLA objects and converting them to compatible FMU types.

The FMU federate is designed adapting the layered simulation architecture approach of Topcu and Oguztuzun [81]. The user interface of the member application is located at the presentation layer. The simulation layer runs the FMU and generate the federate behavior. The FMU is initialized, necessary inputs for FMU are provided from HLA class instances, the model is executed and the model outputs are published to HLA. The communication layer is developed for the RTI communication in order to access the object classes and interactions exchanged in the federation execution.

In contrast to the previous study, this one does not tailor the code generation process, instead it promotes utilization of a standard code generation target, namely FMI. It endeavors to provide a reconfigurable adaptation layer between two standard interfaces, FMI and HLA, in order to support the model-driven practice in fulfilling distant integration requirements. This study acknowledges the value of supporting infrastructures for enabling the pervasion of model-driven practices in simulation enterprise and offers efficiency in simulation systems engineering processes. The reader can find the paper in Appendix B.6.

3.4.2 Testing

DSEEP recommends a three level testing approach: member application testing, integration testing and interoperability testing. The desired outcome is a tested and validated simulation environment. The flexibility and adaptability requirements of objective fidelity assessment approach for flight simulators in the DLR Institute of Flight Systems allowed me to investigate model-driven methodologies at the member application testing level. In collaboration with Wismar University of Applied Sciences, we developed a model-based testing approach for flight simulator objective fidelity evaluation. Regarding this study, I will be presenting two papers.

Ontology for Objective Flight Simulator Fidelity Evaluation

The term flight simulator fidelity is defined as the degree to which a flight simulator matches the characteristics of the real aircraft. Objective fidelity evaluation attacks the fidelity problem with comparison of simulator and the actual flight over some quantitative measures. It is tedious, labor intensive and error prone. There is a solid need for a flexible automated testing methodology not only for executing the tests but also for developing the test cases. Model Based Testing (MBT) is defined as a methodology for automating test case generation from a test specification, also called test model, instead of implementing test cases manually [83]. At first, based on my previous experience about using ontologies as conceptual models and applying model transformations on ontologies, we developed an ontology for objective flight simulator fidelity evaluation, as a metamodel. This section is based on our paper:

Umut Durak, Artur Schmidt and Thorsten Pawletta. 2014. Ontology for objective flight simulator fidelity evaluation. *SNE Simulation Notes Europe* 24, 2, 69-78.

We conducted metamodeling in two levels. An Experimental Frame Ontology (EFO) has been developed as an upper ontology to specify a formal structure for generic simulation test model. Then we constructed Objective Fidelity Evaluation Ontology (OFEO) by extending EFO, to capture domain specific meta test definitions. The ontologies were constructed using OWL [59] in Protégé [58].

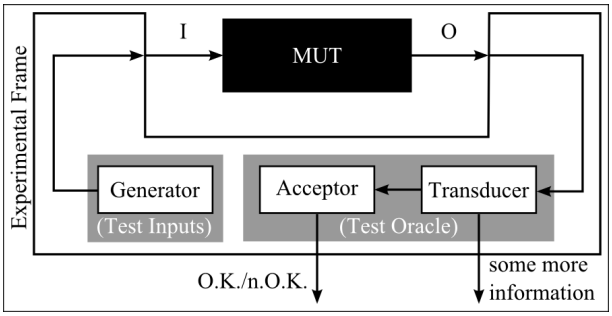


Fig. 3.10. Test case structure based on EF [50]

As presented in Figure 3.10, the structure of a test case is formalized based on Experimental Frame (EF) concept which is developed by Zeigler and his colleagues in the context of Discrete Event System Specification (DEVS). It defines the conditions under which a model is to be examined [84], [85]. The formal specification of the EF is given by a 7-tuple:

$$EF =< T, I, O, C, \Omega_i, \Omega_c, SU >$$

where
T is the time base
I is the set of input variables
O is the set of output variables
C is the set of control variables
Ω_i is the set of admissible input segments
Ω_c is the set of admissible control segment
SU is a set of summary mappings

The implementation of an experimental frame is recommended as a coupled system consisting of a generator, acceptor and a transducer. It is connected to the model, which in our case named as Model Under Test (MUT). The generator produces the test inputs. The set of admissible input segments influences MUTs behavior. Test oracle is composed of the acceptor and transducer. The transducer calculates outcome measures based on output variables and the acceptor decides if an experiment is valid or not.

As depicted in the entity hierarchy of EFO given in Figure 3.11, the structure of a test case is captured in the ontology based on experimental frames. Then each objective validation test case described in ICAO 9625 Manual of Criteria for the Qualification of Flight Training Devices

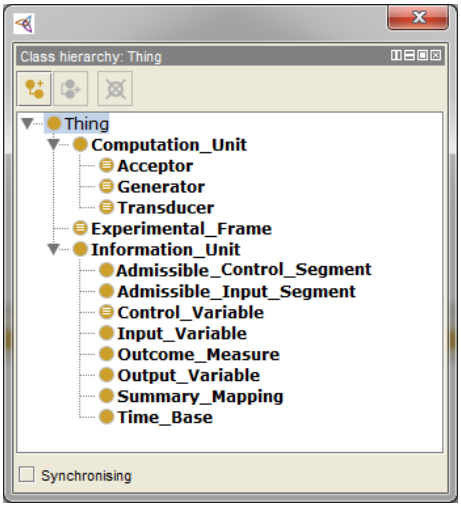


Fig. 3.11. The entity hierarchy of EFO [50]

[86] under performance and handling qualities are specified as an experimental frame in OFEO by extending the definitions in EFO.

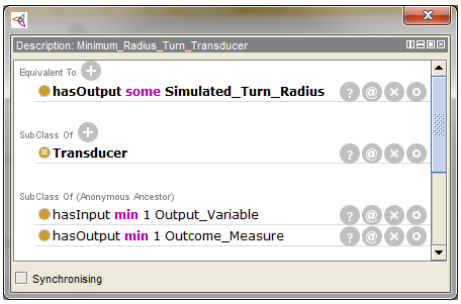


Fig. 3.12. Minimum Radius Turn test transducer description [50]

As an example the transducer of Minimum Turn Radius test description in Protégé is depicted in Figure 3.12. It has a specific output Simulated Turn Radius while it also inherits the properties of a Transducer, thereby, it will be using Output Variables for computing the outcome measure.

This initial attempt provided us with the opportunity to introduce a semantic infrastructure, a metamodel for model-based testing of simulation models based on the fundamental concept of experimental frames from the theory of modeling and simulation. This paper is regarded as an important part of this thesis in that it exemplifies a metamodeling effort that aims to utilize a model-driven practice in simulation systems engineering. The reader can find the paper in Appendix B.7.

Model-Based Testing for Objective Fidelity Evaluation of Engineering and Research Flight Simulators

The initial idea was to use the ontology in OWL and generate test cases using model-transformations. After metamodeling, we started to discuss the possibilities to conduct model transformation from the test model, in that moment described as an ontology to an executable test for testing models in MATLAB/Simulink. One of the options could be developing and OWL to Simulink transformation tool, whereas other option could be using available graph transformation tools. For both cases, we had the concern about the accessibility of model transformation approach by the simulation systems engineers. Then, we converged upon the theory of modeling and simulation and decided to investigate System Entity Structure/Model Base (SES/MB) framework [85] to generate an executable test model from an ontology of all test cases. Rather than using OWL, we utilized SES as an upper ontology in order to specify family of test cases and rather than graph rewriting, we employed SES/MB for model transformation. We published our initial attempt in [51] with a simple case study from the robotics domain. Based on the technical note presented in the previous chapter and following an SES/MB based approach presented by Schmidt et al. [51], we proposed a fullfledged model-based testing for objective fidelity evaluation of engineering and research flight simulators. This section is based on our paper:

> Umut Durak, Artur Schmidt and Thorsten Pawletta. 2015. Model-based testing for objective fidelity evaluation of engineering and research flight simulators. In *AIAA Modeling and Simulation Technologies Conference*, Dallas, TX.

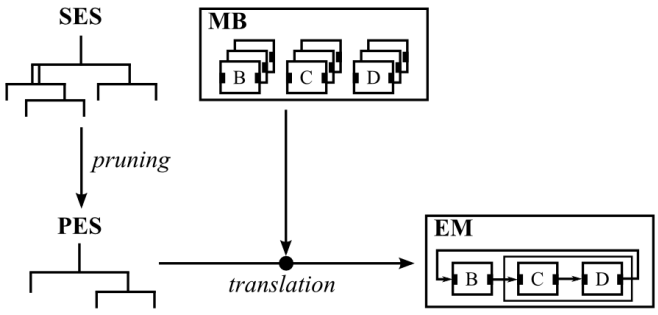


Fig. 3.13. SES/MB framework [85]

The SES is a directed and labeled tree with links to Base Models (BMs) in the Model Base (MB) [87]. It provides a set of elements and axioms to describe system structures. These elements include entity, aspect, specialization and multiple aspect. Entity represents system components. The other elements describe the relationships between entities. While aspect denotes the decomposition relationship of an entity, specialization represents the taxonomy of an entity. And finally, the multiple aspect, specifies a multiple composition relationship. As presented in Figure 3.13, a distinct system structure can be derived from an SES with *pruning* operation. The result is called Pruned Entity Structure (PES). SES Variables are used to specify and configure

the pruning operation. After *pruning*, *translation* operation uses the links to BMs defined in the nodes of a PES to generate an executable simulation model (EM).

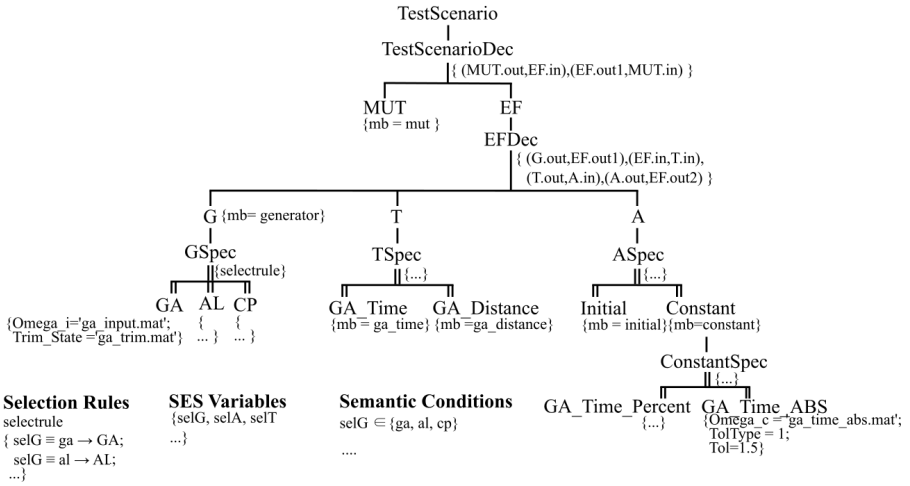


Fig. 3.14. An excerpt from SES for objective fidelity evaluation [52]

We constructed an SES for specifying test case structure based on EFs and extended it with the family of test cases for flight simulator objective fidelity evaluation (Figure 3.14). We then proposed to employ SES/MB for executable test case generation. In the SES, a test scenario is decomposed into an experimental frame (EF) and a model under test (MUT). An EF consists of a generator (G), an acceptor (A) and a transducer (T). The links to the models were specified using mb attribute of the entities. In aspect elements, the model couplings were defined using tuples, such as ($MUT.out$, $EF.in$) which specifies that the output of the MUT is connected with the input of the EF. The entity nodes G, A and T were specialized by using *GSpec*, *ASpec* and *TSpec* for application specific generators, acceptors and transducers. As an example from Figure 3.14, we specialized G to Ground Acceleration (GA), Autopilot Landing (AL) or a Cruise Performance (CP). Selection rules are defined for pruning the test family for a specific test case based of the settings of SES variables. An example can be ga value setting for SES variable $selG$ which implies that GA generator will be used in the specified test configuration.

Figure 3.15 presents on the left hand side a decision free tree, a PES that specifies a test case based on the selection variables:

$$SES\_Vars = \{selG = ga, selT = ga\_time, selA = ga\_time\_abs\}$$

The right hand side is the corresponding executable test case structure that is specified in the PES. As the next step, an overall testing framework that is capable of running test suites that are defined as sets of SES variables was proposed. The infrastructure was prototyped in MATLAB and a sample model base was developed in Simulink. Finally, we managed to generate executable Simulink test cases.

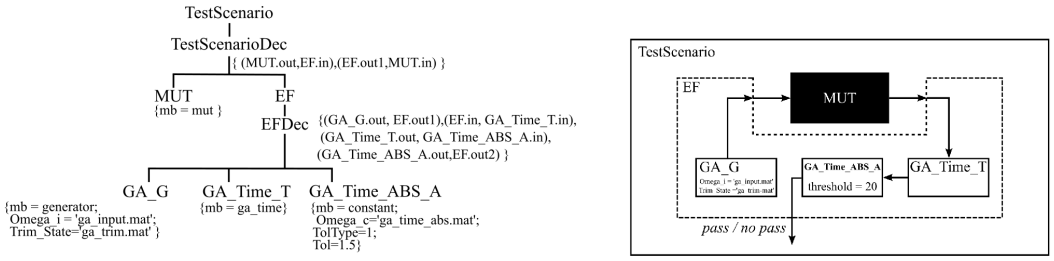


Fig. 3.15. PES and generated executable test [52]

The paper reports a successful adaptation of model-driven methodologies for test case generation in member application testing. The prototype infrastructure implementation in MATLAB/Simulink made the methodology accessible for the simulation systems engineers. Thus, the efficiency of the approach can be exercised by the end users. Following the positive reception of simulation systems engineers, the production SES and MB is being developed to provide a wide range of test cases. The reader can find the paper in Appendix B.8.

3.5 Simulation Installation

Installation is defined in ISO/IEC 12207 [61] as installing the software product that meets the agreed requirements in the target environment. When installation is conducted manually and ad hoc, it is repetitive, labor-intensive, time-consuming and error prone. The automation of an installation requires extensive scripting for various tasks. While scripting for tool automation is commonly employed in software development, especially with the rise of model-based practices, simulation development largely depends upon utilizing graphical modeling approaches. Accordingly, in this section we presented an effort which adapts graphical modeling approaches from simulation modeling and proposes a model-based simulation deployment technique. This section is based on our paper:

> Umut Durak, Anil Ozturk and Mehmet Katircioglu. 2016. Simulation deployment blockset for MATLAB/ Simulink. In *Proceedings of the Symposium on Theory of Modeling & Simulation: DEVS Integrative M&S Symposium*, Pasadena, CA, 630-637.

Deployment is a set of activities from development to release [88]. While it includes the build, deploy, test and release steps in software development, in simulation development it usually consists of a collection of activities, including model checking, Model-in-the-Loop (MIL) testing, code generation, build, Software-in-the-Loop (SIL) testing, deployment, Processor-in-the-Loop (PIL) and Hardware-in-the-Loop (HIL) testing and release. The automation of a simulation deployment pipeline therefore includes triggering model checkers, model compilers and code compilers, configuring test setups, and conducting source control or file system operations. In this study, relying on the competence of simulation systems engineers in Causal Block Diagrams (CBD), we proposed a CBD based approach for automating simulation deployment activities.

We first analyzed targets and the deployment requirements of Indigenous Rotorcraft Simulation (TIRS) of Turkish Aerospace Industries, Inc. (TAI). Then we designed a set of atomic blocks for simulation deployment. Eventually we implemented them as a MATLAB/Simulink blockset.

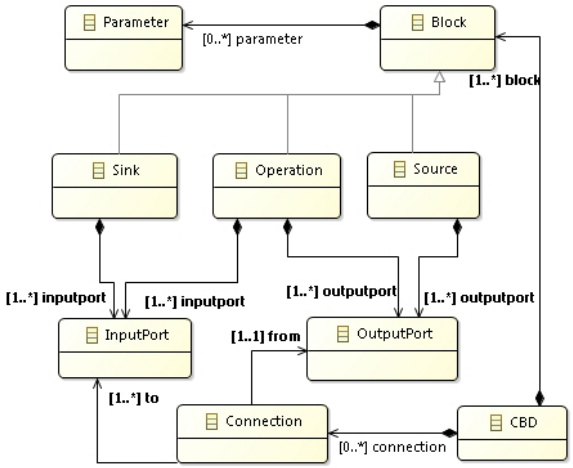


Fig. 3.16. Abstract syntax of Causal Block Diagrams [55]

CBD is a general formalism that is widely utilized for modeling of causal and continuous-time systems [89]. They are largely employed in control systems design and referred as the common language of industry in the embedded systems design process [90]. The abstract syntax of CBD for the purpose of this study is developed as depicted in Figure 3.16, and Simulink is utilized as the concrete syntax. *Source*, *Sink* and *Operation* are types of *Block*. *Source* generates signals and *Sink* consumes signals. *Operation* applies algebraic, trigonometric, logical or, even further algorithmic manipulations on the signals. At last, *Block* may possess *Parameter*s that configure their operations. *Block* may have *InputPorts* and/or *OutputPorts*.

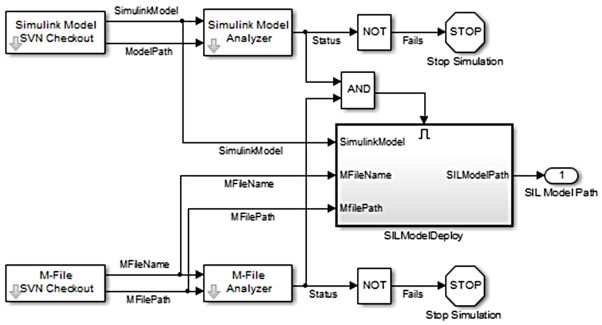


Fig. 3.17. Top level SIL deployment automation model [55]

Totally 13 atomic blocks were designed and implemented some of which are Simulink Model SVN Checkout, Simulink Model Analyzer, S-Function Generator, Simulink Code Generator and

SIL Model Deploy. They can be used with all other standard and user defined Simulink blocks. A sample deployment automation model is developed for the software-in-the-loop testing of TIRS. The top level Simulink model for the sample case is given in Figure 3.17.

This study demonstrated the applicability of model-driven practices out of development process area. Further anecdotal evidence provided promising insight that modelers pose positive tendencies to employ graphical modeling for further process automation. This paper is regarded as influential in this thesis with its potential to push modeling practices from development towards the automation of engineering processes. The reader can find the paper in Appendix B.9.

3.6 Simulation Maintenance

Maintenance is also one of the process areas that is overlooked in DSEEP. ISO/IEC 12207 [61] defines maintenance as supporting a delivered product. This includes problem and modification analysis, modification implementation, acceptance and migration. In this section, two articles that promote model-driven methodologies for the maintenance of simulations will be presented.

3.6.1 Simulation Modernization

Architecture Driven Modernization (ADM) has been proposed by OMG as a model-based approach to software maintenance in which knowledge extracted from the software assets are captured in models, and model transformations are recommended as the means of modification [91]. I found the ADM approach very promising for modernizing legacy simulation assets by transforming embedded implicit knowledge in the legacy assets to target architectures. The formalisms that are required for representing software assets during reverse engineering are specified by OMG in the Knowledge Discovery Metamodel (KDM) [92]. KDM is well accepted by the software engineering community and further efforts have been spent on developing supporting toolsets, such as MoDisco [93]. However, the diversity in methodologies and approaches to specify simulation modeling assets prohibits KDM from providing adequate meta definitions to capture knowledge in simulations. Therefore, in this study I proposed a methodology and an extension to KDM for model-based reverse engineering of simulations. This section is based on the paper:

> Umut Durak. 2015. Extending the Knowledge Discovery Metamodel for architecture-driven simulation modernization. *Simulation* 91, 12, 1052-1067.

Modernization of simulation assets is a well-known problem. Many legacy simulation tools and technologies are now either no longer supported or have considerably lost their market today. Various studies attacked the problem with promoting wrappers and communication methods with the legacy simulations, rather than proposing a structured, reengineering based modernization approach [94, 95, 96, 97]. However in software engineering, long lasting efforts to define a standard modernization process for software systems have led to ADM.

KDM is a metamodel that represents software, its elements, associations and operational environments [98]. While KDM is well applicable for knowledge discovery in simulation software, the knowledge discoverable with the available KDM packages will barely encompass anything about simulation modeling. As presented in Figure 3.18, an extension to KDM is proposed in this study by adapting SES. *Simulation Model* provides user with an abstract elements for

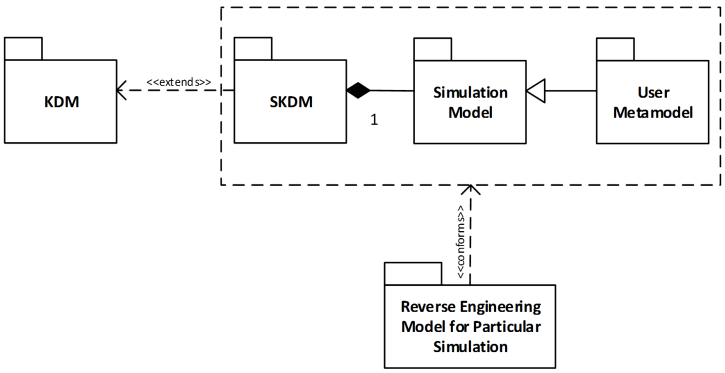


Fig. 3.18. Simulation Knowledge Discovery Metamodel [53]

constructing an SES. It needs to be extended with concrete metamodel elements for specific simulation knowledge discovery requirements.

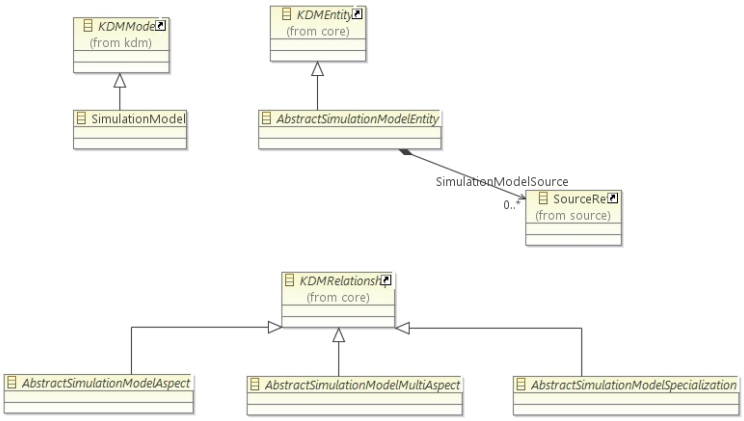


Fig. 3.19. Simulation Model inheritance diagram [53]

The elements Simulation Model package is presented in Figure 3.19. *AbstractSimulationModelEntity* is an abstract model entity. It introduces the Entity node in SES and is derived from *KDMEntity*. *AbstractSimulationModelAspect* is an abstract aspect relationship and represents the Aspect node in SES. *AbstractSimulationModelSpecialization* is an abstract specialization relationship and corresponds Specialization node from SES. *AbstractSimulationModelMultiAspect* is an abstract multi-aspect relationship and is based on the definition of Multi-Aspect node in SES. These three classes are derived from *KDMRelationship*.

The paper presents three case studies that extends Simulation Model package for particular purposes. In the first example, a simulation knowledge discovery metamodel is presented based on Discrete-event Modeling Ontology (DeMO) [44]. The second example introduces another simulation model discovery metamodel for Modelica which is a domain specific language for physical systems simulation [99]. In the last example, a simulation knowledge discovery metamodel is

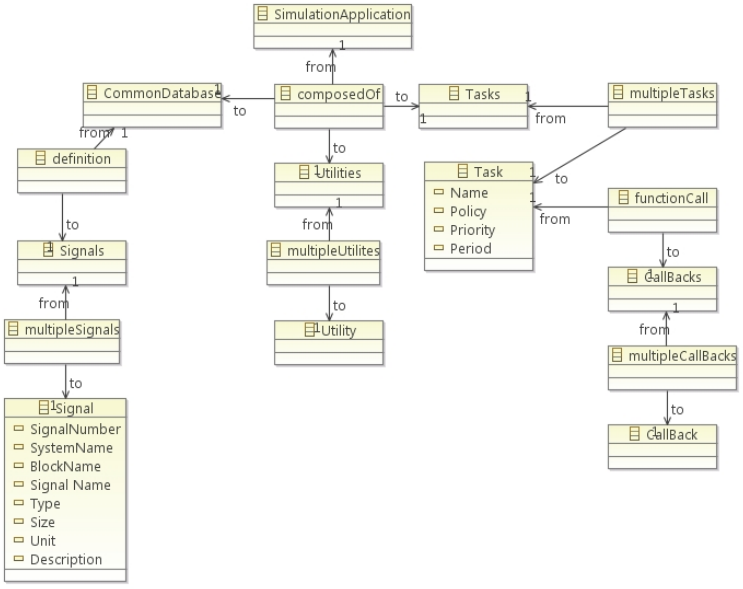


Fig. 3.20. 2Simulate Model class diagram [53]

introduced for real-time distributed simulation based on 2Simulate, the real-time distributed simulation framework of DLR. The class diagram of this metamodel is given in Figure 3.20. This metamodel is then employed to conduct knowledge discovery from a legacy helicopter simulation asset.

This paper presented the utilization of model-driven methodologies in reverse engineering of simulations for the first time. Thus, it enabled a simulation modernization approach aligned with ADM. In particular, by using the case study that presents the reverse engineering of a legacy flight simulator member application, its application is unfolded. This article is regarded as significant in this thesis since it extends the utilization of model-driven practices from forward engineering to reverse engineering of simulations. The reader can find the paper in Appendix B.10.

3.6.2 Model Refactoring

Refactoring has been widely employed in software development as an evolutionary modernization technique that targets altering the structure of the artifact incrementally in order to achieve a better quality, while keeping its behavior unchanged. Along with the well-established research on code refactoring [100], there have been some efforts at model refactoring [37, 101, 102, 103, 104, 105]. Unfortunately, while they present a particular model-driven approach, all of these efforts target at MATLAB/Simulink. Even though there are various simulation modeling environments that support different graphical modeling languages, refactoring has not been addressed for any of these tools and environments. In this study, I proposed an in-place model transformations for refactoring in Scilab/Xcos. This section is based on the paper:

Umut Durak. 2016. Pragmatic model transformations for refactoring in Scilab/Xcos. *International Journal of Modeling, Simulation, and Scientific Computing* 7,1, Article 1541004, 23 pages.

In this study, first the metamodel of Scilab/Xcos was constructed based on the scs_m data structure [106] that Scilab stores the Xcos model data. The aim was to make its abstract syntax explicitly available as the baseline of the model transformation. Based on the metamodel, an in-place model transformation approach is promoted. It comprises matching a precondition pattern (Left Hand Side) in the model and replacing it with the outcome pattern (Right Hand Side) in the same model [107].

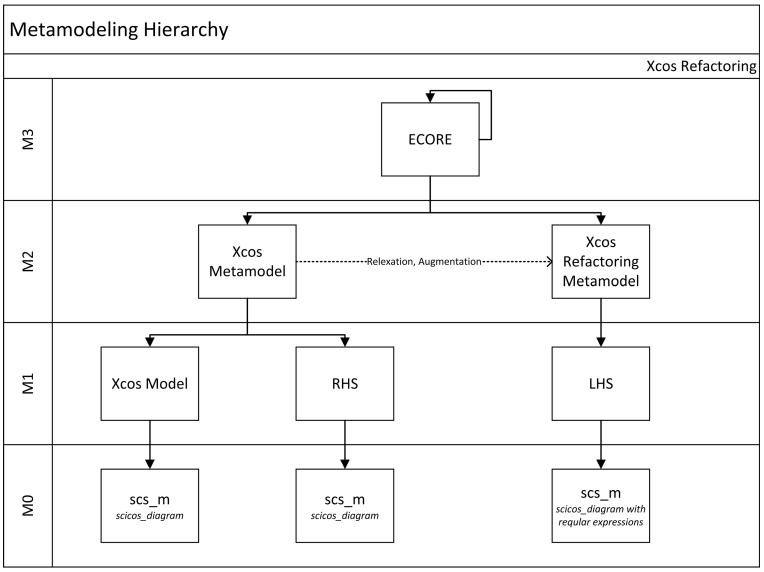


Fig. 3.21. Metamodeling hierarchy for model refactoring [54]

Following Kühne and his colleagues, pattern specification metamodels was obtained by subjecting the original language metamodel to relaxation, augmentation and modification [108]. In concrete level, search patterns are proposed to be developed using the scs_m structure augmented with regular expressions to define the constraints. The Xcos metamodel is simplified for refactoring purposes as a relaxation regarding the suitable fields for constraint definition, and all the data types of the parameter values are specified as strings in order to enable the application of regular expressions. The RHS outcome patterns are regarded as model fragments and completeness is not required from them, but they are specified using the same structure with the model conforming to Xcos metamodel. The proposed metamodeling hierarchy is presented in Figure 3.21.

For the definition of precondition and outcome model, the concrete syntax as Xcos diagrams is exploited. The constraint specifications that consists regular expressions as the declarative constraints are set to any parameter in LSH scs_m structure with scripting. For the transformations, Scilab scripting level Application Programming Interface (API) is proposed. API not only pro-

vides atomic model transformation functions for finding, adding, deleting and replacing a block or a link, but also serves composite model transformation functions for finding, adding, deleting and replacing a subdiagram. Finally, the proposed approach and infrastructure is exemplified with two case studies that consist non-trivial refactoring scenarios.

Other than being the pioneer effort about model refactoring in Scilab/Xcos and presenting a model-driven approach to the model refactoring problem, the commitment of the approach presented in this paper is to make the model-transformations accessible for the simulation systems engineer with providing native Scilab model transformation function rather than employing external model transformation tools and languages. The reader can find the final accepted version of the paper in Appendix B.11.

4

Outlook and Future Directions

This thesis is built upon the lengthy experience of the simulation community with modeling and modeling languages. Its motivation can be summarized as making the model-driven methodologies, namely modeling, metamodeling and model transformations across technical spaces, a practice of engineering of simulation systems.

Concerning the wide range of methodologies and approaches used in simulation modeling, this thesis argues that unified languages for modeling are not adequate. Consequently, it is hard to come up with a common and widely accepted Model-Driven Engineering methodology that covers all the engineering processes. Additionally, in contrast to the systems engineering of software intensive systems where Model-Driven Engineering is better established, the shared understanding of the concepts and steps of the simulation systems engineering life cycle is still relatively weak.

The thesis acknowledges the virtues of model-driven methodologies and promotes their application in the various steps of simulation systems engineering without proposing an integrated model-driven engineering process. This approach is termed as Model-Based Simulation Systems Engineering.

The thesis begins with a recent position paper that proposes an ontology for simulation systems engineering towards creating a shared conceptualization. With this paper, we would like to prompt a discussion about a common understanding of the simulations system engineering life cycle, aiming for a more mature modeling and simulation discipline. Our follow-up efforts in this direction include a book chapter "An Index to the Body of Knowledge of Simulation Systems Engineering" in an upcoming title, *The Profession of Modeling and Simulation*, edited by Andreas Tolk and Tuncer Oren to be published by John Wiley & Sons, Inc. In order to establish engineering practices, such as those in this thesis, it is necessary to provide the connections between the practices and the process. Therefore, I consider the simulation engineering life cycle to be a prerequisite for disseminating Model-Based Simulation Systems Engineering.

Various applications of modeling, metamodeling and model transformations are presented at particular steps of simulation systems engineering. Making use of modeling expertise of simulation community, we pushed the modeling practice from development to other process areas to enable automation. We exercised this idea in simulation installation and collected positive feedback. There are various simulation assets that can be represented as models. Metamodeling is, therefore, crucial. In various papers that are presented in this thesis, metamodels and metamodeling are employed. Model transformations are regarded as the least known model-driven practice in the modeling and simulation community. In this thesis, there are number of papers that present their implementation for particular purposes. We applied them for generating member application

designs and simulation data exchange models from conceptual models and conceptual scenarios; or executable scenarios from conceptual scenarios. Additionally, we employed them for model refactoring. There is also a paper in which we address integration challenges using model-to-text transformations. We interpreted System Entity Structure / Model Base framework from the theory of modeling and simulation as a model transformation technique and proposed a model-based testing approach. We also emphasized the importance of supporting infrastructures for the establishment of model-driven practices in the paper that adapts Functional Mock-up Units to High Level Architecture federates.

In this effort, the applicability of model-driven practices in the steps of simulation systems engineering was under investigation along with their promise of increasing productivity, efficiency and effectiveness. The presented papers bear witness to the success of model-driven methodologies in the engineering of simulation systems. Based on the definition of Model-Based Engineering, which advocates pragmatic utilization of model-driven practices in engineering process areas, this thesis is the initial attempt to introduce Model-Based Simulation Systems Engineering.

The aspiration of this effort is to make model-driven practices a part of the standard simulation systems engineering toolset. The upcoming challenge is to make these practices accessible for actors of simulation systems engineering, particularly to the development/integration team. Here, it is necessary to mention that the foundations of the model-driven practices, including graph theory, graph rewriting and transformations, formal methods, and language design, are not usually within the capability set of this team. Cross-layer approaches that will guide the practitioners of model-driven practices through the application of underlying fundamental techniques are of utmost importance. Accordingly, supporting model-driven practices within the feature set of modeling and simulation tools is critical. Therefore, we need to reinterpret the requirements of modeling and simulation tools from a Model-Based Simulation Systems Engineering perspective. Lastly, we need to work on creating a strategy to integrate model-based practices in our modeling and simulation engineering, not only in modeling and simulation or computer science programs, but further in engineering curricula.

References

1. P. K. Davis and R. H. Anderson, "Improving the composability of DoD models and simulations," *The Journal of Defense Modeling and Simulation: Applications, Methodology, Technology*, vol. 1, no. 1, pp. 5–17, 2004.
2. T. Ören, "The many facets of simulation through a collection of about 100 definitions," *SCS M&S Magazine*, vol. 2, no. 2, pp. 82–92, 2011.
3. T. Ören, "A critical review of definitions and about 400 types of modeling and simulation," *SCS M&S Magazine*, vol. 2, no. 3, pp. 142–151, 2011.
4. T. I. Ören, "Uses of simulation," *Principles of Modeling and Simulation: A Multidisciplinary Approach*, pp. 153–179, 2009.
5. O. Balci, "Guidelines for successful simulation studies (tutorial session)," in *Proceedings of the 22nd Winter Simulation Conference*, pp. 25–32, IEEE, 1990.
6. T. Ören and L. Yilmaz, "Synergies of simulation, agents, and systems engineering," *Expert Systems with Applications*, vol. 39, no. 1, pp. 81–88, 2012.
7. C. Haskins, *Systems Engineering Handbook: A Guide for System Life Cycle Processes and Activities Version 3.2.2.* No. INCOSE-TP-2003-002-03.2.2., INCOSE, 2011.
8. T. I. Ören and L. Yilmaz, "Synergy of systems engineering and modeling and simulation," in *Proceedings of the 2006 International Conference on Modeling and Simulation-Methodology, Tools, Software Applications (M&S MTSA)*, pp. 10–17, 2006.
9. U. Durak and T. Ören, "Towards an ontology for simulation systems engineering," in *Proceedings of the 48th Annual Simulation Symposium*, SCS, 2016.
10. IEEE, "IEEE Recommended Practice for Distributed Simulation Engineering and Execution Process (DSEEP)," *IEEE Std 1730-2010*, 2011.
11. IEEE, "IEEE Recommended Practice for High Level Architecture (HLA) Federation Development and Execution Process (FEDEP)," *IEEE Std 1516.3-2003*, 2003.
12. J. Bézivin, "On the unification power of models," *Software & Systems Modeling*, vol. 4, no. 2, pp. 171–188, 2005.
13. M. Brambilla, J. Cabot, and M. Wimmer, *Model-Driven Software Engineering in Practice*. Morgan & Claypool Publishers, 2012.
14. O. Topçu, U. Durak, H. Oğuztüzün, and L. Yilmaz, *Distributed Simulation: A Model Driven Engineering Approach*. Springer, 2016.
15. INCOSE, *Vision 2020*. No. INCOSE-TP-2004-004-02, INCOSE, 2007.
16. C. Atkinson and T. Kühne, "Model-driven development: a metamodeling foundation," *IEEE Software*, vol. 20, no. 5, pp. 36–41, 2003.
17. D. Gasevic, D. Djuric, and V. Devedic, *Model driven engineering and ontology development*. Springer Science & Business Media, 2009.
18. OMG, "OMG Unified Modeling Language (OMG UML™) version 2.5," 2015.
19. OMG, "OMG Meta Object Facility (MOF) core specification version 2.5," 2015.
20. OMG, "Object Management Group Model Driven Architecture (MDA) MDA guide rev. 2.0," 2014.

21. D. Steinberg, F. Budinsky, E. Merks, and M. Paternostro, *EMF: Eclipse Modeling Framework*. Pearson Education, 2008.
22. OMG, "OMG Systems Modeling Language (OMG SysML™) version 1.3," 2015.
23. C. A. Petri, "Communication with automata: Volume 1 supplement 1," tech. rep., DTIC Document, 1966.
24. H. M. Paynter, *Analysis and design of engineering systems*. MIT press, 1961.
25. A. Tolk, "Avoiding another green elephant - a proposal for the next generation HLA based on the model driven architecture," in *Fall Simulation Interoperability Workshop*, SISO, 2002.
26. S. Parr and R. Keith-Magee, "Making the case for MDA," in *Fall Simulation Interoperability Workshop*, 2003.
27. J. De Lara and H. Vangheluwe, "Atom^3: A tool for multi-formalism and meta-modelling," in *Proceedings of the 5th International Conference on Fundamental Approaches to Software Engineering*, pp. 174–188, Springer-Verlag, 2002.
28. P. Mosterman and H. Vangheluwe, "An introduction to computer automated multi-paradigm modeling," *Simulation*, vol. 80, no. 7, pp. 3–16, 2004.
29. H. Vangheluwe and J. De Lara, "Computer automated multi-paradigm modelling for analysis and design of traffic networks," in *Proceedings of the 36th Winter Simulation Conference*, pp. 249–258, IEEE, 2004.
30. G. Özhan and H. Oguztüzün, "Model-integrated development of HLA-based field artillery simulation," in *European Simulation Interoperability Workshop*, SISO, 2006.
31. D. Çetinkaya and H. Oğuztüzün, "A metamodel for the HLA object model," in *Proceedings of the 20th European Conference on Modeling and Simulation*, pp. 207–213, 2006.
32. O. Topçu, M. Adak, and H. Ouztüzün, "A metamodel for federation architectures," *ACM Transactions on Modeling and Computer Simulation (TOMACS)*, vol. 18, no. 3, 2008.
33. M. Adak, O. Topçu, and H. Oguztüzün, "Model-based code generation for HLA federates," *Software: Practice and Experience*, vol. 40, no. 2, pp. 149–175, 2010.
34. J. N. Duarte and J. de Lara, "O D i M: A model-driven approach to agent-based simulation," in *Proceedings of the 23rd European Conference on Modelling and Simulation*, pp. 158–165, 2009.
35. S. Mittal and S. A. Douglass, "From domain specific languages to DEVS components: Application to cognitive M&S," in *Proceedings of the 2011 Symposium on Theory of Modeling & Simulation: DEVS Integrative M&S Symposium*, pp. 256–265, SCS, 2011.
36. U. B. Ighoroje, O. Maïga, and M. K. Traoré, "The DEVS-driven modeling language: Syntax and semantics definition by meta-modeling and graph transformation," in *Proceedings of the 2012 Symposium on Theory of Modeling and Simulation - DEVS Integrative M&S Symposium*, SCS, 2012.
37. J. Denil, P. J. Mosterman, and H. Vangheluwe, "Rule-based model transformation for, and in Simulink," in *Proceedings of the Symposium on Theory of Modeling & Simulation-DEVS Integrative*, SCS, 2014.
38. J. Ledet, A. Teran-Somohano, Z. Butcher, L. Yilmaz, A. E. Smith, H. Oğuztüzün, O. Dayıbaş, and B. K. Görür, "Toward model-driven engineering principles and practices for model replicability and experiment reproducibility," in *Proceedings of the Symposium on Theory of Modeling & Simulation-DEVS Integrative*, SCS, 2014.
39. S. Mittal and B. P. Zeigler, "DEVS unified process for web-centric development and testing of system of systems," tech. rep., DTIC Document, 2008.
40. T. Iba, Y. Matsuzawa, and N. Aoyama, "From conceptual models to simulation models: Model driven development of agent-based simulations," in *9th Workshop on Economics and Heterogeneous Interacting Agents*, 2004.
41. E. Guiffard, D. Kadi, J.-P. Mochet, and R. Mauget, "CAPSULE: application of the MDA methodology to the simulation domain," in *European Simulation Interoperability Workshop*, SISO, 2006.
42. A. D'Ambrogio, D. Gianni, J. L. Risco-Martín, and A. Pieroni, "A MDA-based approach for the development of DEVS/SOA simulations," in *Proceedings of the 2010 Spring Simulation Multiconference*, SCS, 2010.
43. D. Cetinkaya, A. Verbraeck, and M. D. Seck, "MDD4MS: A model driven development framework for modeling and simulation," in *Proceedings of the 2011 Summer Computer Simulation Conference*, SCSC '11, pp. 113–121, SCS, 2011.

44. G. A. Silver, J. A. Miller, M. Hybinette, G. Baramidze, and W. S. York, "DeMO: an ontology for discrete-event modeling and simulation," *Simulation*, vol. 87, no. 9, pp. 747–773, 2011.
45. Ö. Özdikiş, U. Durak, and H. Oğuztüzün, "User-guided transformations for ontology based simulation design," in *Proceedings of the 2009 Summer Computer Simulation Conference*, pp. 75–82, SCS, 2009.
46. Ö. Özdikiş, U. Durak, and H. Oğuztüzün, "Tool support for transformation from an OWL ontology to an HLA object model," in *Proceedings of the 3rd International ICST Conference on Simulation Tools and Techniques*, ICST, 2010.
47. U. Durak, O. Topçu, R. Siegfried, and H. Oğuztüzün, "Scenario development: A model-driven engineering perspective," in *Proceedings of the 2014 International Conference on Simulation and Modeling Methodologies, Technologies and Applications (SIMULTECH)*, pp. 117–124, IEEE, 2014.
48. F. Yılmaz, U. Durak, K. Taylan, and H. Oğuztüzün, "Adapting Functional Mockup Units for HLA-compliant distributed simulation," in *Proceedings of the 10th International Modelica Conference*, pp. 247–257, 2014.
49. T. Gerlach, U. Durak, and J. Gotschlich, "Model integration workflow for keeping models up to date in a research simulator," in *Proceedings of the 2014 International Conference on Simulation and Modeling Methodologies, Technologies and Applications (SIMULTECH)*, pp. 125–132, IEEE, 2014.
50. U. Durak, A. Schmidt, and T. Pawletta, "Ontology for objective flight simulator fidelity evaluation," *SNE Simulation Notes Europe*, vol. 24, no. 2, pp. 69–78, 2014.
51. A. Schmidt, U. Durak, C. Rasch, and T. Pawletta, "Model-based testing approach for MATLAB/Simulink using system entity structure and experimental frames," in *Proceedings of the Symposium on Theory of Modeling & Simulation: DEVS Integrative M&S Symposium*, pp. 69–76, SCS, 2015.
52. U. Durak, A. Schmidt, and T. Pawletta, "Model-based testing for objective fidelity evaluation of engineering and research flight simulators," in *AIAA Modeling and Simulation Technologies Conference*, AIAA, 2015.
53. U. Durak, "Extending the Knowledge Discovery Metamodel for architecture-driven simulation modernization," *Simulation*, vol. 91, no. 12, pp. 1052–1067, 2015.
54. U. Durak, "Pragmatic model transformations for refactoring in Scilab/Xcos," *International Journal of Modeling, Simulation, and Scientific Computing*, vol. 7, no. 1, p. 1541004, 2016.
55. U. Durak, A. Ozturk, and M. Katircioglu, "Simulation deployment blockset for MATLAB/Simulink," in *Proceedings of the Symposium on Theory of Modeling & Simulation: DEVS Integrative M&S Symposium*, SCS, 2016.
56. T. R. Gruber, "Toward principles for the design of ontologies used for knowledge sharing," *International Journal of Human-Computer Studies*, vol. 43, no. 5, pp. 907–928, 1995.
57. R. Mizoguchi, "Ontological engineering: Foundation of the next generation knowledge processing," in *Web Intelligence*, pp. 44–57, Springer, 2001.
58. H. Knublauch, R. W. Fergerson, N. F. Noy, and M. A. Musen, "The Protégé OWL plugin: An open development environment for semantic web applications," in *The Semantic Web–ISWC 2004*, pp. 229–243, Springer, 2004.
59. S. Bechhofer, "OWL: Web Ontology Language," in *Encyclopedia of Database Systems*, pp. 2008–2009, Springer, 2009.
60. S. Lohmann, S. Negru, F. Haag, and T. Ertl, "Visualizing ontologies with VOWL," *Semantic Web*, pp. 1–21, Preprint.
61. IEEE, "ISO/IEC/IEEE Standard for Systems and Software Engineering - Software Life Cycle Processes," *IEEE STD 12207-2008*, 2008.
62. R. Siegfried, A. Laux, M. Rother, D. Steinkamp, G. Herrmann, J. Lüthi, and M. Hahn, "Scenarios in military (distributed) simulation environments," in *Spring Simulation Interoperability Workshop*, SISO, 2012.
63. R. Siegfried, H. Oguztüzün, U. Durak, A. Hatip, G. Herrmann, P. Gustavson, and M. Hahn, "Specification and documentation of conceptual scenarios using base object models (BOMs)," in *Proceedings of the 2013 Spring Simulation Conference*, 2013.

64. D. Franceschini, R. Franceschini, R. Burch, R. Sherrett, and J. Abbott, "Specifying scenarios using the military scenario definition language," in *Simulation Interoperability Workshop*, SISO, 2004.
65. F. Jouault, F. Allilaire, J. Bézivin, I. Kurtev, and P. Valduriez, "ATL: a QVT-like transformation language," in *Companion to the 21st ACM SIGPLAN Symposium on Object-Oriented Programming Systems, Languages, and Applications*, pp. 719–720, ACM, 2006.
66. A. Agrawal, "Great: A metamodel based model transformation language," *Institute for Software Integrated Systems (ISIS), Vanderbilt University*, 2003.
67. OMG, "Meta object facility (MOF) 2.0 Query/View/Transformation specification," 2008.
68. Y. Pan, G. Xie, L. Ma, Y. Yang, Z. Qiu, and J. Lee, "Model-driven ontology engineering," in *Journal on Data Semantics VII*, pp. 57–78, Springer, 2006.
69. U. Durak, H. Oguztuzun, and S. K. Ider, "An ontology for trajectory simulation," in *Proceedings of the 2006 Winter Simulation Conference*, pp. 1160–1167, IEEE, 2006.
70. U. Durak, G. Mahmutyazicioglu, and H. Oguztuzun, "Domain analysis for reusable trajectory simulation," in *European Simulation Interoperability Workshop*, pp. 303–312, SISO, 2005.
71. U. Durak, H. Oguztuzun, and S. K. Ider, "Ontology-based trajectory simulation framework," *Journal of Computing and Information Science in Engineering*, vol. 8, no. 1, p. 014503, 2008.
72. U. Durak, H. Oguztuzun, and S. K. Ider, "Ontology-based domain engineering for trajectory simulation reuse," *International Journal of Software Engineering and Knowledge Engineering*, vol. 19, no. 08, pp. 1109–1129, 2009.
73. A. Ledeczi, M. Maroti, A. Bakay, G. Karsai, J. Garrett, C. Thomason, G. Nordstrom, J. Sprinkle, and P. Volgyesi, "The generic modeling environment," in *Workshop on Intelligent Signal Processing*, 2001.
74. C. F. Koksal Algin, "Ontology driven development for HLA federates," Master's thesis, Middle East Technical University, Ankara, Turkey, 2010.
75. MathWorks, "Simulink Coder Generate C and C++ code from Simulink and Stateflow models." `http://www.mathworks.com/products/simulink-coder/`. Accessed: 2016-31-01.
76. D. Allerton, *Principles of flight simulation*. John Wiley & Sons, 2009.
77. J. Gotschlich, T. Gerlach, and U. Durak, "2simulate: a distributed real-time simulation framework," in *Workshop der ASIM/GI-Fachgruppen STS und GMMS*, 2014.
78. MathWorks, "Simulink Coder Target Language Compiler." `http://www.mathworks.com/help/pdf_doc/rtw/rtw_tlc.pdf`. Accessed: 2016-01-06.
79. T. Blochwitz, M. Otter, J. Åkesson, M. Arnold, C. Clauss, H. Elmqvist, M. Friedrich, A. Junghanns, J. Mauss, D. Neumerkel, *et al.*, "Functional Mockup Interface 2.0: The standard for tool independent exchange of simulation models," in *9th International Modelica Conference*, pp. 173–184, The Modelica Association, 2012.
80. ESA, "SMP 2.0 handbook," *EGOS-SIM-GEN-TN-0099*, 2005.
81. O. Topçu and H. Oğuztüzün, "Layered simulation architecture: a practical approach," *Simulation Modelling Practice and Theory*, vol. 32, pp. 1–14, 2013.
82. F. D. Group, "Functional Mock-up Interface for Model Exchange and Co-Simulation," *FMI 2.0*, 2014. Accessed: 2016-01-06.
83. J. Zander, I. Schieferdecker, and P. J. Mosterman, *Model-based testing for embedded systems*. CRC press, 2011.
84. B. P. Zeigler, *Multifacetted modelling and discrete event simulation*. Academic Press Professional, Inc., 1984.
85. B. P. Zeigler, H. Praehofer, and T. G. Kim, *Theory of modeling and simulation: integrating discrete event and continuous complex dynamic systems*. Academic press, 2000.
86. ICAO, "Manual criteria for the qualification of flight training devices," *ICAO 9625*, 2009.
87. B. P. Zeigler and P. E. Hammonds, *Modeling & simulation-based data engineering: introducing pragmatics into ontologies for net-centric information exchange*. Academic Press, 2007.
88. J. Humble and D. Farley, *Continuous delivery: reliable software releases through build, test, and deployment automation*. Pearson Education, 2010.
89. E. Posse, J. De Lara, and H. Vangheluwe, "Processing causal block diagrams with graphgrammars in atom3," in *European Joint Conference on Theory and Practice of Software (ETAPS), Workshop on Applied Graph Transformation (AGT)*, pp. 23–34, 2002.

90. B. Denckla and P. J. Mosterman, "Formalizing causal block diagrams for modeling a class of hybrid dynamic systems," in *Decision and Control, 2005 and 2005 European Control Conference. CDC-ECC'05. 44th IEEE Conference on*, pp. 4193–4198, IEEE, 2005.
91. R. Pérez-Castillo, I. G. R. de Guzmán, and M. Piattini, "Architecture-driven modernization," *Modern Software Engineering Concepts and Practices: Advanced Approaches*, 2010.
92. R. Pérez-Castillo, I. G.-R. De Guzman, and M. Piattini, "Knowledge Discovery Metamodel - ISO/IEC 19506: A standard to modernize legacy systems," *Computer Standards & Interfaces*, vol. 33, no. 6, pp. 519–532, 2011.
93. H. Bruneliere, J. Cabot, F. Jouault, and F. Madiot, "Modisco: a generic and extensible framework for model driven reverse engineering," in *Proceedings of the IEEE/ACM International Conference on Automated Software Engineering*, pp. 173–174, ACM, 2010.
94. M. T. Radosevic, J. Hensen, and A. T. M. Wijsman, "Distributed building performance simulation – a novel approach to overcome legacy code limitations," *HVAC&R Research*, vol. 12, no. S1, pp. 621–640, 2006.
95. E. L. White and J. M. Pullen, "Adapting legacy computational software for xmsf," in *Fall Simulation Interoperability Workshop*, SISO, 2003.
96. M. Sonntag, S. Hotta, D. Karastoyanova, D. Molnar, and S. Schmauder, "Using services and service compositions to enable the distributed execution of legacy simulation applications," in *Towards a Service-Based Internet*, pp. 242–253, Springer, 2011.
97. L. W. Lacy, *Interchanging Discrete event simulation Process Interaction Models using the Web Ontology Language-OWL*. PhD thesis, University of Central Florida Orlando, Florida, 2006.
98. OMG, "Knowledge Discovery Meta-Model (KDM) version 1.3," 2011.
99. P. Fritzson and V. Engelson, "Modelica – a unified object-oriented language for system modeling and simulation," in *ECOOP98 Object-Oriented Programming*, pp. 67–90, Springer, 1998.
100. T. Mens and T. Tourwé, "A survey of software refactoring," *IEEE Transactions on Software Engineering*, vol. 30, no. 2, pp. 126–139, 2004.
101. Q. M. Tran, B. Wilmes, and C. Dziobek, "Refactoring of Simulink diagrams via composition of transformation steps," in *International Conference on Software Engineering Advances*, pp. 140–145, 2013.
102. C. Amelunxen, E. Legros, A. Schürr, and I. Stürmer, "Checking and enforcement of modeling guidelines with graph transformations," in *Applications of Graph Transformations with Industrial Relevance*, pp. 313–328, Springer, 2007.
103. I. Stürmer and D. Travkin, "Automated transformation of MATLAB Simulink and Stateflow models," in *Proceedings of the 4th Workshop on Object-oriented Modeling of Embedded Real-time Systems*, pp. 57–62, 2007.
104. H. Giese, M. Meyer, and R. Wagner, "A prototype for guideline checking and model transformation in Matlab/Simulink," in *Proceedings of the 4th International Fujaba Days*, pp. 56–60, 2006.
105. T. Farkas, C. Hein, and T. Ritter, "Automatic evaluation of modelling rules and design," in *Second Workshop From Code Centric to Model Centric Software Engineering: Practices, Implications and ROI*, 2006.
106. S. L. Campbell, J.-P. Chancelier, and R. Nikoukhah, *Modeling and Simulation in Scilab/Scicos*. Springer, 2006.
107. K. Czarnecki and S. Helsen, "Feature-based survey of model transformation approaches," *IBM Systems Journal*, vol. 45, no. 3, pp. 621–645, 2006.
108. T. Kühne, G. Mezei, E. Syriani, H. Vangheluwe, and M. Wimmer, "Explicit transformation modeling," in *Models in Software Engineering*, pp. 240–255, Springer, 2009.

Part II

Appendix

A

List of Own Publications and Description of my Contribution

1. Umut Durak and Tuncer Oren. 2016. Towards an ontology for simulation systems engineering. In *Proceedings of the 48th Annual Simulation Symposium*, Pasadena, CA, 163-170.

   Contribution: Oren contributed to the paper by providing an far-reaching introduction that presents the evolution of modeling and simulation with regards to the synergies with other disciplines and introduced the term *simulation systems engineering*. Then I elaborated the simulation systems engineering concept and processes based on DSEEP. While the paper aims to start a discussion in the modeling and simulation community towards developing an ontology for a shared conceptualization of the discipline, it also serves this thesis to provide a framework in which that model-driven methodologies will be placed. Throughout the thesis, I extensively make use of DSEEP concepts and take DSEEP as a reference for the simulation systems engineering process. Both co-authors contributed to the work in equal terms.

2. Umut Durak, Okan Topcu, Robert Siegfried and Halit Oguztuzun. 2014. Scenario development: A model-driven engineering perspective. In *Proceedings of the 4th International Conference on Simulation and Modeling Methodologies, Technologies and Applications (SIMULTECH)*, SCITEPRESS, Vienna, Austria, 75-82.

   Contribution: In this paper, I collaborated with Siegfried to align the work with his scenario development approach, with Topcu in constructing the process and Oguztuzun for the model-driven methodologies. My contribution to this very paper was more than the half. The conceptual scenario metamodel, conceptual scenario development case study and sample model transformations were constructed by myself.

3. Ozer Ozdikis, Umut Durak and Halit Oguztuzun. 2009. User-guided transformations for ontology based simulation design. In *Proceedings of the 2009 Summer Computer Simulation Conference*, SCS, Istanbul, Turkey, 75-82.

   Contribution: In this paper, the development of model transformation techniques and tools were done within the scope of Ozdikis's master thesis. Each co-author contributed this effort about the same amount, where my contributions can be listed as methodology development, definition of mappings between ontology constructs and target metamodel constructs.

4. Ozer Ozdikis, Umut Durak and Halit Oguztuzun. 2010. Tool support for transformation from an OWL ontology to an HLA Object Model. In *Proceedings of the 3rd International ICST Conference on Simulation Tools and Techniques*, ICST, Malaga, Spain.

   Contribution: In this paper, out of evenly shared contributions of co-authors to the effort, mine were the development of the methodology and construction of mappings.

5. Torsten Gerlach, Umut Durak and Jurgen Gotschlich. 2014. Model integration workflow for keeping models up to date in a research simulator. In *Proceedings of the 4^{th} International Conference on Simulation and Modeling Methodologies, Technologies and Applications (SIMULTECH)*, SCITEPRESS, Vienna, Austria, 125-132.

   Contribution: Gerlach defined the requirements for model integration for research flight simulators and initiated the effort, I then contributed by proposing the model-driven approach which addresses the model integration challenges as a part of code generation. Gotschlich conducted a significant amount of the implementation, while I supported him by developing the scripts that automate the build process. Each co-author contributed the paper about the same amount.

6. Faruk Yilmaz, Umut Durak, Koray Taylan and Halit Oguztuzun. 2014. Adapting Functional Mockup Units for HLA-compliant distributed simulation. In *Proceedings of the 10th International Modelica Conference*, Lund, Sweeden, 247-257.

   Contribution: This study was conducted as a part of the MOKA (*Turkish*: **MO**del **K**ullanim **A**ltyapisi, *English*: Model Utilization Infrastructure) Research Project funded by the Turkish Ministry of National Defence, Undersecretariat for Defence Industries. MOKA aimed to adapt FMI technologies in the model-driven engineering processes of Roketsan Missiles Inc. It was a university-industry research grant. While I was the principal investigator of MOKA from the industry side, Oguztuzun led the project at university. Yilmaz conducted the development as a part of his masters thesis that Oguztuzun and I co-advised, and Taylan provided us with valuable management insights. All co-authors contributed to the study in equal measure.

7. Umut Durak, Artur Schmidt and Thorsten Pawletta. 2014. Ontology for objective flight simulator fidelity evaluation. *SNE Simulation Notes Europe* 24, 2, 69-78.

   Contribution: In this paper, I conducted the ontology development, Schmidt and Pawletta contributed with their expertise in experimental frames. My contribution to this paper was about the half.

8. Umut Durak, Artur Schmidt and Thorsten Pawletta. 2015. Model-based testing for objective fidelity evaluation of engineering and research flight simulators. In *AIAA Modeling and Simulation Technologies Conference*, Dallas, TX.

   Contribution: In this paper, I employed the methodology that we developed in [51] and the tool infrastructure from Schmidt and Pawletta in order to address flexibility and adaptability

requirements for a testing approach in objective fidelity evaluation of engineering and research flight simulator. All co-authors contributed to this paper in equal terms.

9. Umut Durak, Anil Ozturk and Mehmet Katircioglu. 2016. Simulation deployment blockset for MATLAB/ Simulink. In *Proceedings of the Symposium on Theory of Modeling & Simulation: DEVS Integrative M&S Symposium*, Pasadena, CA, 630-637.

   Contribution: My contribution to this study was about the half. I developed the methodology and specified the blocks, block interfaces and block features. Ozturk and Katircioglu contributed to the study with implementation and the case study.

10. Umut Durak. 2015. Extending the Knowledge Discovery Metamodel for architecture-driven simulation modernization. *Simulation* 91, 12, 1052-1067.

11. Umut Durak. 2016. Pragmatic model transformations for refactoring in Scilab/Xcos. *International Journal of Modeling, Simulation, and Scientific Computing* 7,1, Article 1541004, 23 pages.

B

Copies of Published Articles

B.1 Towards an Ontology for Simulation Systems Engineering

Umut Durak and Tuncer Oren. 2016. Towards an ontology for simulation systems engineering. In *Proceedings of the 48th Annual Simulation Symposium*, Pasadena, CA, 163-170.

Towards an Ontology for Simulation Systems Engineering

Umut Durak
DLR Institute of Flight Systems
Lilienthalplatz 7
38108 Braunschweig, Germany
umut.durak@dlr.de

Tuncer Ören
University of Ottawa
School of Electrical Engineering and
Computer Science
P.O. Box 450, Stn A
Ottawa, Ontario, Canada, K1N 6N5
oren@eecs.uOttawa.ca

ABSTRACT
Although modeling and simulation and systems engineering are distinct fields, the increasing complexity of the systems of interest lead us to two emerging disciplines, namely simulation systems engineering and simulation-based systems engineering. This paper is about simulation systems engineering which is simulation enhanced by systems engineering. On one hand, it can be perceived as a natural step in the evolution of the discipline of simulation, on the other hand, the concepts of simulation systems engineering, the tools and the methodologies, the form and the content of work products, the types of data stores and the information flows currently show a great variety. Notwithstanding, simulation systems engineering requires a shared conceptualization for coordination, cooperation and integration. There have been previous efforts that contributed to the creation of a common vocabulary. Extending them, this paper proposes to develop an ontology for simulation systems engineering, which will specify not only a common but also an explicit vocabulary. The simulation systems engineering process captured in an ontology is envisioned to help creating a common understanding about the concepts of simulation systems engineering and streamlining information exchange within and among simulation systems engineering activities.

Author Keywords
Simulation Systems Engineering; Ontology; DSEEP

ACM Classification Keywords
I.6.0 SIMULATION AND MODELING (General).

1. INTRODUCTION

To appreciate the evolution of simulation, consideration of its synergies with several disciplines is in order. In this article, first, we clarify the concept of first-, second- and high-order synergies. Section 2 is about simulation systems engineering where the evolution of simulation to simulation systems engineering as well as the simulation systems engineering lifecycle are clarified. Section 3 is dedicated to the ontology of simulation systems engineering where the following issues are covered: ontology as a common vocabulary, simulation systems engineering ontology, and the representation of this ontology. Section 4 covers our conclusion and our future activities on the subject.

1.1 Synergies

Synergies play an important role in the evolution of disciplines. As outlined in Figure 1, two disciplines may mutually contribute to the enhancement of each other. Hence "a" may contribute to "b" and vice versa. Each contribution is a first-order synergy.

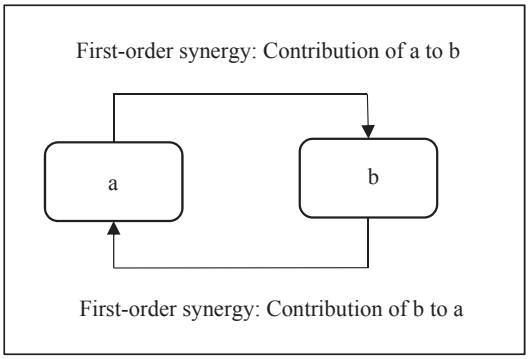


Figure 1 First-order synergies of a and b

Further, there are also second-order synergies as depicted in Figure 2. In a typical second-order synergy of three disciplines "a," "b," and "c," for example, "b" may enhance "c" (first-order synergy); however, if "b" is already enhanced by "a," then this contribution "a" to the enhancement of "c" is second-order synergy. Likewise higher order synergies also exist. In this way, disciplines contribute to each other and evolve.

SpringSim-ANSS, 2016 April 3-6, Pasadena, CA, USA

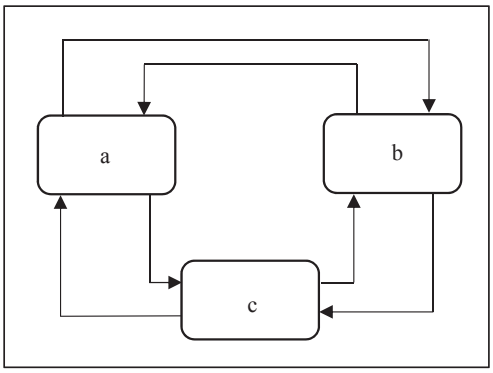


Figure 2 Second-order synergies of a, b and c

Figure 3 represents several other 1st, 2nd, and high-order synergies among modeling and simulation, computerization and software engineering, artificial intelligence and software agents, system theories, soft computing, and systems engineering. Most of these synergies were elaborated elsewhere by Ören and Yilmaz [23] and are not repeated here.

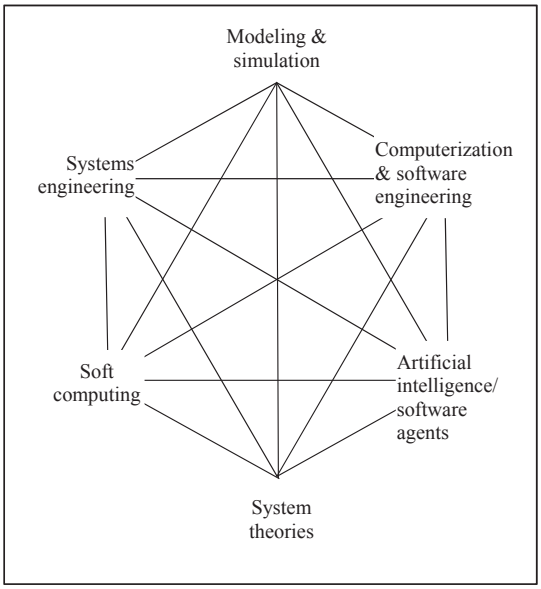


Figure 3 Synergies between simulation and other disciplines [19]

2. SIMULATION SYSTEMS ENGINEERING

2.1 Evolution of Simulation to Simulation Systems Engineering

The importance of a proper definition of simulation was emphasized on several occasions [20, 21]. Simulation is the use of dynamic models to perform *experimentations* or to *gain experience* or for *entertainment*. However, simulation can also be conceived from an abstract knowledge processing point of view. From this point of view, *simulation is a model-based experiential knowledge generation process* [18, 19]. As outlined in Figure 4, this abstract point of view is useful to comprehend the evolution of simulation as a series of aspects of sophistication of simulation by its synergies with several disciplines. (Contributions of simulation to other disciplines – albeit very important and interesting – are not covered here.)

At the beginning (*aspect 1*), simulation was non-computational; thought experiments [3], scale models, and sandbox models were used.

Aspect 2 (computerized simulation) was reached by the use of computers. At the beginning, analog and later hybrid computers were used. With the advent of digital computers, programming, and later software engineering contributed to make simulation a versatile tool. Petascale and exascale computers offer simulationists tremendous opportunities. The ubiquitousness of computers such as wearable and implantable computers as well as cloud computing may open new vistas for simulationists too.

In the nineteen sixties, systems theories started to contribute to make simulation even more powerful. Hence *aspect 3,* theory-based simulation, started a new era of formal simulation with important implications such as model-based simulation that promotes utilizing formal models which also started model-based approaches in other disciplines such as software engineering. Simulation engineering marked an important level of advanced simulation studies. The maturity of the simulation field entailed simulation-based science and engineering

Aspect 4 is intelligent (or cybernetic) simulation. It was made possible by the synergies of simulation with artificial (or machine) intelligence and later with software agents. Both of the possibilities offer benefits of the synergies, such as contribution of simulation to agents (and AI) and contributions of agents (and AI) to simulation [19].

The *5th aspect* is soft computing simulation, which is the result of contribution of soft computing to simulation. It involves neural network simulation, fuzzy simulation (for non-numerical computation in simulation) and swarm simulation.

The *6th aspect* is simulation systems engineering which results from the contribution of systems engineering to simulation. The other aspect, the contribution of simulation to systems engineering, i.e., simulation-based systems engineering – though very important – is not elaborated here.

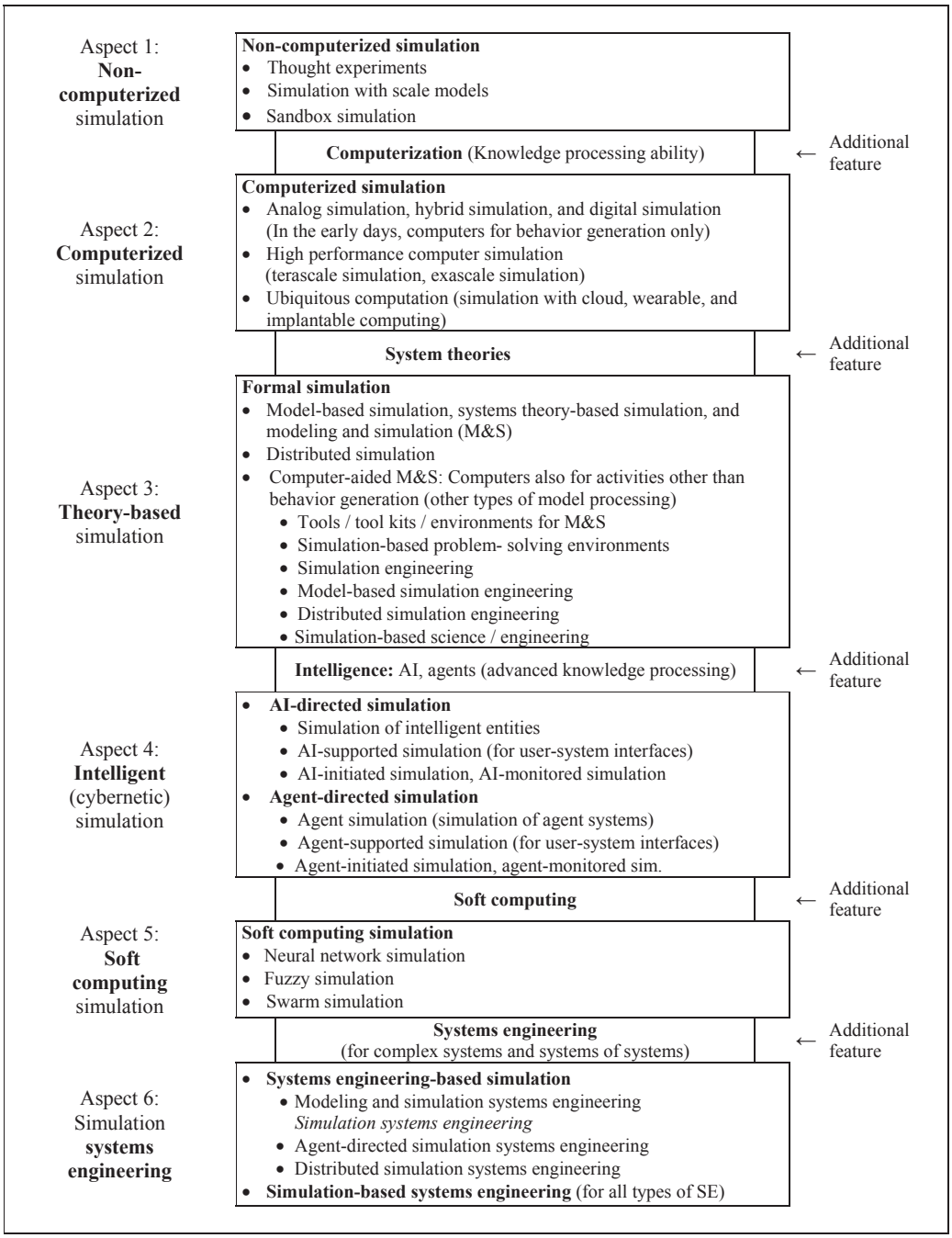


Figure 4 Evolution of simulation through a sequence of aspects of simulation (based on Ören [19])

2.2 Synergies of Simulation and Systems Engineering

International Council of Systems Engineering (INCOSE) defines systems engineering as an interdisciplinary approach and means to enable the realization of successful systems [25]. The increasing maturity and mutual contributions of both disciplines, simulation and systems engineering, offer the opportunity of their synergistic relations as outlined in Figure 4. What is important is that both disciplines contribute to the enhancement of each other while maintaining their identities.

Contribution of simulation to a discipline "x" is called *simulation-based x*. Already, simulation-based science, simulation-based engineering as well as many other simulation-based disciplines are important examples of contributions of simulation to other disciplines, making simulation a powerful infrastructure for them. Similarly, systems engineering cannot and should not be considered without its relation with simulation (Figure 5). Hence, simulation-based systems engineering.

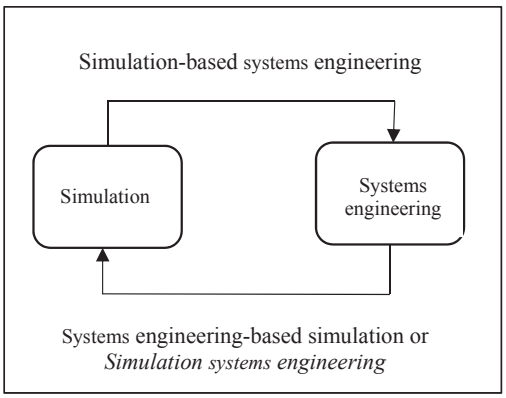


Figure 5 Synergy of simulation and systems engineering

Similarly, the contribution of systems engineering to simulation empowers simulation especially for large and complex system (and systems of systems) simulation. Hence, we have systems engineering-based simulation or *simulation systems engineering*, for short. Many issues of social systems need to be studied by simulation systems engineering. Accordingly, simulation-based social systems engineering needs to be matured to benefit from simulation systems engineering and from social systems engineering. This way, root causes of problems in complex social systems can be properly analyzed and fact-based long-term solutions can be recommended.

2.3 Simulation Systems Engineering Lifecycle, DSEEP and Beyond

Simulation systems engineering executes an interdisciplinary process for developing, maintaining and employing simulations for experimenting and gaining experience about systems of interest. The early studies for defining the life cycle process of modeling and simulation studies trace back to the early 1990s when Balci introduced a life cycle for a simulation study [1]. Later efforts that target defining a process for federation development, particularly for distributed simulations that utilize High Level Architecture lead to an IEEE standard, namely IEEE Std 1516.3-2003, IEEE Recommended Practice for High Level Architecture (HLA) Federation Development and Execution Process (FEDEP) [9]. It has been well accepted as a starting framework for tailoring an end-to-end process for the development and execution of HLA federations.

In 2007, the Simulation Interoperability Standards Organization (SISO) FEDEP Product Development Group (PDG) decided to generalize FEDEP in order to come up with an engineering process for all types of distributed simulation development and execution. In 2010, it was published as IEEE Std 1730-2010 IEEE Recommended Practice for Distributed Simulation Engineering and Execution Process (DSEEP) [10].

The DSEEP-recommended process starts with defining simulation environment objectives. At this step, objectives of the simulation experiment are set forth by the consensus among stakeholders, such as users, sponsors, and developers. The requirements engineering study is introduced as the second step. Scenario development, conceptual modeling, and requirements definition are carried out in this step by the development team. Then the simulation environment is designed at the third step. The members or the participants of the distributed simulation are selected. The members to be reused are identified as well as the ones to be newly developed. The requirements are also allocated to the members in this step. The third step is completed with a development and implementation plan for the simulation environment. In the fourth step, following the development of a data exchange model and the simulation environment agreement, new members as well as the modifications to the existing members are implemented. Integration and test of the simulation environment are the main goals of the fifth step. Interoperability requirements are verified in this step via integration and testing. In the sixth step, the distributed simulation is executed. The process concludes with the seventh step in which data analysis and evaluation are carried out.

DSEEP states that the implementations of this seven-step process can show a great variety. It further introduces activity descriptions with their inputs and outputs and a list of representative recommended tasks. Data flow diagrams are employed by DSEEP in order to graphically represent the product flow among the activities, as well as to introduce the data stores. Additionally, five roles are defined, namely user/sponsor, simulation environment manager, development/ integration team, verification and validation agent and accreditation/acceptance agent.

While DSEEP contributed in great amount towards the standardization of the simulation systems engineering lifecycle process and although data flow diagrams help a lot in presenting the information in a structured way, it is still far from providing an unambiguous, systematic and structured explanation of the shared terms and vocabulary for coordination, cooperation and integration of simulation systems engineering processes. As an example, whilst DSEEP defines the inputs and outputs of the activities, it provides no further classification of the inputs and outputs, i.e. if they are products or information. The naming of the information items and products is usually not enough to create an explicit specification. Info on available resources or supporting resources, for example, are regarded as inadequate definitions for creating a shared vocabulary.

3. ONTOLOGY FOR SIMULATION SYSTEMS ENGINEERING

3.1. Ontology as a Common Vocabulary

In philosophy, the term ontology has the meaning of a systematic explanation of existence. In artificial intelligence, it was first Neches et al. who defined ontology as the basic terms and relations comprising the vocabulary of a topic area as well as the rules for combining terms and relations to define extensions to the vocabulary [16]. Later in 1993, Gruber's definition "Ontology is explicit specification of conceptualization" [7] became famous. Then this definition is explained by Struder and colleagues. They introduced "conceptualization" as an abstract model of some phenomenon in the world which identifies the relevant concepts of that phenomenon and "explicit" as type of concepts used, and the constraints on their use are explicitly defined [24].

Mizoguchi listed the merits of ontologies as: a) common vocabulary, b) explication of what has often been left implicit, c) systematization of knowledge, d) standardization and e) meta-model functionality [15].

The applications of ontologies are classified in four main categories: Neutral authoring, ontology as specification, common access to information and ontology-based search [6]. Application ontologies in the engineering domain mostly dropped into the first three categories. The early efforts on developing engineering ontologies were in the 1990s. The PhysSys [2] was one of the first engineering ontologies. It is based on system dynamics theory that is practiced in engineering modeling, simulation and design. The PhysSys was developed to formally define how design engineers or the end users of Computer Aided Engineering (CAE) systems understand their domain and to provide a foundation for the conceptual schema for data structuring in engineering databases, libraries and other CAE information systems. The ideas captured in PhysSys are regarded here as a baseline for the development of a library of reusable models for engineering and design.

It was 2004 when Miller and his colleagues introduced how ontologies can be helpful in modeling and simulation [14]. In their paper, they proposed that ontologies may be useful for concept browsing, querying and navigating, simulation service discovery, simulation component repository development, hypothesis testing, platform or machine independent simulation specification and shared conceptual framework development. Since then, there have been many efforts to utilize ontologies for modeling and simulation. The ACM Digital Library returns about 200 matches for a search with "ontology" as keyword only in Society for Modeling & Simulation International (SCS) conference proceedings since that paper was published.

Ören used ontologies in his Modeling and Simulation Body of Knowledge (M&S BoK) [22] and Comprehensive and Integrative View of M&S (Big Picture) [19] studies in order to provide an ontology-based dictionary for the terms to show also their logical relationships.

This study refers to the merit of ontologies as the common vocabulary and makes use of explication of what has often been left implicit and of systematization of knowledge. Therefore, rather than a modeling language like Business Process Model and Notation (BPMN)[17], it promotes utilization of ontologies. It proposes to extend the ontology-based dictionary of modeling and simulation and to provide the simulation community with an ontology of simulation systems engineering. Thus the concepts of the simulation systems engineering process that emerged from the synergy between systems engineering and modeling and simulation can be explicitly specified.

3.2 Simulation Systems Engineering Ontology

This section is an initial attempt to develop an ontology for simulation systems engineering. This attempt is based on the simulation systems engineering process that has been described in DSEEP. In addition, remarks and discussions about the classifications and specifications of the terms in DSEEP will be put forward as the direction into which the presented initial endeavor needs to evolve in order to achieve its goals.

In virtue of their popularity, Protégé [8] is used as the ontology development environment and ontologies are developed using the Ontology Web Language (OWL). Since introducing OWL would exceed the scope of this paper, the reader is kindly recommended to read ref [4] for the detailed description.

The top level entities of the simulation systems engineering ontology are adopted from the basic concepts of DSEEP. As depicted in Figure 6, they are Activity, Data Store, Information, Product, Role, Step and Task. Each entity is then further inherited to its child entities. In

the first place, steps and data stores can be discussed. Steps proposed by DSEEP have their roots from FEDEP.

The seven steps of simulation systems engineering are captured in the ontology. The data stores mentioned in the standard are authoritative resources, data dictionaries, M&S repositories, scenario databases, simulation data exchange models and other resources. They are captured as the child entities of *Data_Store*. But the authors would like to stress the necessity for further elaboration of the taxonomy of data stores.

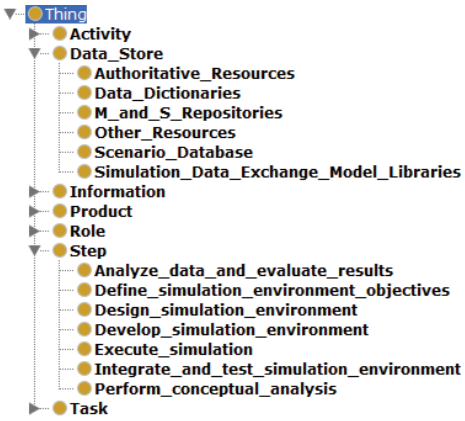


Figure 6 The hierarchy of the simulation systems engineering ontology

The basic structure of DSEEP proposes that steps are composed of activities and activities are composed of tasks. The inputs and outcomes are defined for steps and activities, but not for tasks. The relations among the entities defined in this structure can be captured by object properties in the ontology. The initial list of proposed object properties is as given in Figure 7. *activityInputs*, *activityOutcomes* are suggested for the input and outputs of the activities, likewise *stepInputs* and *stepOutcomes* for the inputs and outputs of steps. *recommendedTasks* capture the representative recommended tasks of the activities and analogously *containsActivities* specify the activities of the steps. Lastly *participateIn* declares which roles participate in which activities.

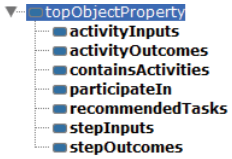


Figure 7 Top level object properties.

Figure 8 presents the lists of *Product* and *Information* entities that are extracted from DSEEP. The items that are input to the overall process are named as *Information* and the items that are created with the activities are named as *Product*. Here the authors would like to stress the requirement to develop a more descriptive taxonomy for the products of the activities of the simulation systems engineering.

Figure 8 Product and Information entities

Five roles described in DSEEP are captured as seven entities in the ontology as depicted in Figure 9, but further elaboration is required here to come up with a more comprehensive taxonomy of roles in the simulation systems engineering process.

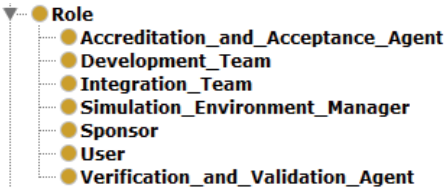


Figure 9 Role entities

The steps of the simulation systems engineering are proposed to be specified using property restrictions. Figure 10 presents the specification of *Define Simulation Environment Objectives* as a *Step* which *containsActivities*:

a) *Identify_user_and_sponsor_needs*,
b) *Develop_objectives* and
c) *Conduct_initial_planning*.

Further it requires particular *stepInputs*:

a) *Information_on_available_resources*,
b) *Program_goals*, and
c) *Existing_domain_descriptions*.

And it concludes with particular *stepOutcomes*:

a) *Objectives_statement* and
b) *Initial_planning_documents*.

Also the activities need specifications using property restrictions.

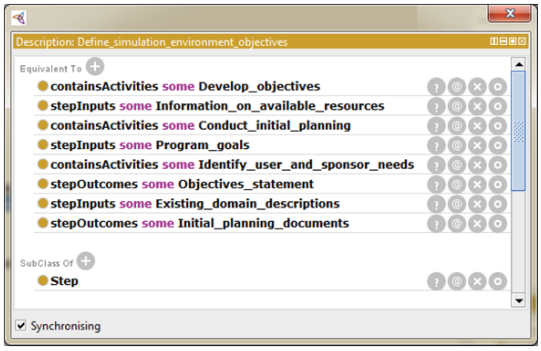


Figure 10 Define Simulation Environment Objectives entity

3.3 Representation of the Ontology

The interpretability and understandability of the constructed ontology by human readers is as important as its machine processability. While machine processibility is envisioned as the enabler for the future ontology-based simulation process integration research, human readability is regarded as the quality that will guarantee the creation of common and shared conceptualizations.

It is reported that although various visualization methods have been proposed for OWL ontologies in the past decade, none of them evolved into the de facto standard [5].

Visual notation for OWL ontologies (VOWL) is a recent effort that defines a visual language for the user-oriented representation of OWL ontologies [12]. It targets at representing ontologies in a way that not only ontology experts, but the domain experts can also understand them easily.

VOWL can be utilized within the Protégé ontology development environment as a plugin, namely ProtégéVOWL [11]. There is also a web application which is called WebVOWL [13].

As introduced in ref [12] in a detailed way, the building blocks of VOWL are graphical primitives and color schemes. Classes are pictured as circles. The solid lines connecting the classes represent the properties with their domain and range axioms. Arrowheads are employed in order to point to the class or the datatype that is designated as the range. The rectangles are used to represent the property labels and datatypes. For cardinality constraints and labels, text fields are employed. The dotted lines indicate subclass relations. The color scheme of VOWL enables easy distinction of different elements. In the recommended scheme, the general color is light blue, whereas datatype properties are light green. Figure 11 presents an excerpt from the representation of the simulation systems engineering ontology using VOWL as an example.

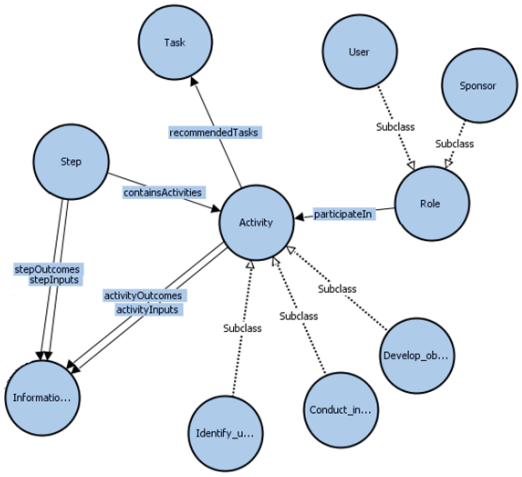


Figure 11 An excerpt from the representation of the simulation systems engineering ontology using VOWL

VOWL with its configurable simple representation strategy that targets non-ontology experts and its web-based representation capabilities is regarded as promising to be utilized for sharing SSE ontology over the web.

4. CONCLUSION

This paper proposes an ontology development effort for the simulation systems engineering process in order to create a common shared conceptualization of products, information exchange, data stores, roles, steps, activities and tasks of the process. While it introduces DSEEP as the baseline for such an effort, it also points out the points where further discussion and elaboration is required to come up with a well-accepted taxonomy. On one side, this study envisions enabling interoperability and cooperation within the simulation systems engineering process with machine-readable shared vocabulary, on the other hand, it discusses the ways to make the ontology accessible for the non-ontology experts using ontology visualization tools.

An initial ontology construction is carried out using OWL in Protégé and VOWL is employed for the visualization. OWL and Protégé are a widely used, mature and accepted language respectively environment for ontology development. VOWL language and tools, on the contrary, are quite new. The maturity and feature set of the available toolset are expected to be enhanced. Further filtering (show/hide), navigation and query capabilities are required.

The ontology will soon be made available for the simulation community over the web, with mechanisms that will enable collaborative editing and maintenance.

ACKNOWLEDGMENT
The authors appreciate Wulf Mönnich for the careful review and editing. It certainly improved the presentation of this article.

REFERENCES

1. Balci, O. Guidelines for successful simulation studies. *Proceedings of the 22nd Winter Simulation Conference*, IEEE Press (1990), 25-32.
2. Borst, R., Akkermans, J., Pos, A., Top, J. The PhysSys Ontology for Physical Systems. *The 9th International Workshop on Qualitative Reasoning*, Amsterdam, Netherlands, 1995.
3. Brown, J.R. Thought Experiments, Stanford Encyclopedia of Philosophy. http://plato.stanford.edu/entries/thought-experiment/. As of 24 February 2016.
4. Bechhofer, S., van Harmelen, F., Hendler, J., Horrocks, I., McGuinness, D.L., Patel-Schneider, P.F., Stein, L.A. OWL web ontology language reference. Edited by Dean, M., Schreiber, G. W3C Recommendation, 2004. http://www.w3.org/TR/owl-ref/. As of 24 February 2016.
5. Dudáš, M., Zamazal, O., Svátek, V. Roadmapping and navigating in the ontology visualization landscape. *The 19th International Conference on Knowledge Engineering and Knowledge Management*, Linköping, Sweden, 2014.
6. Falbo, R.A., Guizzardi, G., Duarte, K.C. An Ontological Approach to Domain Engineering. *International Conference on Software Engineering and Knowledge Engineering*, Ischia, Italy, 2002.
7. Gruber, T. Toward Principles for the Design of Ontologies Used for Knowledge Sharing. *Int. Journal of Human-Computer Studies*, 43 (1995), 907-928.
8. Knublauch, H., Horridge, M., Musen, M., Rector, A., Stevens, R., Drummond, N., Lord, P., Noy, N., Seidenberg, J., Wangl, H. The Protégé OWL Experience. *OWLED*, Galway, Ireland, 2005.
9. IEEE Std 1516.3-2003. *IEEE Recommended Practice for High Level Architecture (HLA) Federation Development and Execution Process (FEDEP)*, 2003.
10. IEEE Std 1730TM-2010. *IEEE Recommended Practice for Distributed Simulation Engineering and Execution Process (DSEEP)*, 2010.
11. Lohmann, S., Negru, S., Bold, D. The ProtégéVOWL Plugin: Ontology Visualization for Everyone. *The Semantic Web: ESWC 2014 Satellite Events*, Crete, Greece, 2014.
12. Lohmann, S., Negru, S., Haag, F., Ertl, T. Visualizing Ontologies with VOWL. *Semantic Web Journal*, to appear. http://www.semantic-web-journal.net/system/files/swj892.pdf. As of 24 February 2016.
13. Lohmann, S., Link, V., Marbach, E., Negru, S. WebVOWL: Web-based Visualization of Ontologies. *Knowledge Engineering and Knowledge Management,* Springer International Publishing (2014), 154-158.
14. Miller, J.A., Baramidze, G.T., Sheth, A.P., Fishwick, P.A. Investigating Ontologies for Simulation Modeling. *37th Annual Simulation Symposium*, Arlington, VA, 2004.
15. Mizoguchi, R. Ontological Engineering: Foundations of the Next Generation Knowledge Processing. *Web Intelligence 2001*, Maebashi City, Japan, 2001.
16. Neches, R., Fikes, R.E., Finin, T., Gruber, T.R., Senator, T., Swartout, W.R. Enabling Technology for Knowledge Sharing, *AI Magazine*, 12,3(1991), 36-56.
17. OMG, Business Process Model and Notation (BPMN) Version 2.0.2, Object Management Group, 2013. http://www.omg.org/spec/BPMN/2.0.2/ As of 24 February 2016.
18. Ören, T.I. Maturing Phase of the Modeling and Simulation Discipline. *ASC - Asian Simulation Conference 2005 (The Sixth International Conference on System Simulation and Scientific Computing (ICSC'2005))*, Beijing, P.R. China, 2005.
19. Ören, T.I. Modeling and Simulation: A Comprehensive and Integrative View. *Agent-Directed Simulation and Systems Engineering.* Wiley-Berlin (2009), 3- 36.
20. Ören, T.I. The Many Facets of Simulation through a Collection of about 100 Definitions. *SCS M&S Magazine*, 2, 2 (2011), 82-92.
21. Ören, T.I. A Critical Review of Definitions and About 400 Types of Modeling and Simulation. *SCS M&S Magazine*, 2, 3 (2011), 142-151.
22. Ören, T.I. A Basis for a Modeling and Simulation Body of Knowledge Index: Professionalism, Stakeholders, Big Picture, and Other BoKs. *SCS M&S Magazine*, 2, 1 (2011), 40-48.
23. Ören, T., Yilmaz, L. Synergies of simulation, agents, and systems engineering. *Expert Systems with Applications*, 39, 1 (2012), 81-88.
24. Struder, R., Benjamins, V.R., Fensel, D. Knowledge Engineering: Principles and Methods, *IEEE Transactions on Knowledge and Data Engineering.* 25, 1-2(1998), 161-167.
25. Walden, D.D., Roedler, G.J., Fosberg, K.K., Hamelin, R.D., Shortell, T.M. (Eds.), *System Engineering Handbook - A Guide for System Life Cycle Processes and Activities*, Wiley, Hoboken, New Jersey, 2015.

B.2 Scenario Development: A Model-Driven Engineering Perspective

Umut Durak, Okan Topcu, Robert Siegfried and Halit Oguztuzun. 2014. Scenario development: A model-driven engineering perspective. In *Proceedings of the 4th International Conference on Simulation and Modeling Methodologies, Technologies and Applications (SIMULTECH)*, SCITEPRESS, Vienna, Austria, 75-82.

Scenario Development: A Model-Driven Engineering Perspective

Umut Durak[1], Okan Topçu[2], Robert Siegfried[3] and Halit Oğuztüzün[4]
[1]*Institute of Flight Systems, German Aerospace Center (DLR), Lilienthalplatz 7, Braunschweig, Germany*
[2]*Department of Computer Engineering, Naval Science and Engineering Institute, Istanbul, Turkey*
[3]*Aditerna GmbH, Riemerling, Germany*
[4]*Department of Computer Engineering, Middle East Technical University, Ankara, Turkey*
umut.durak@dlr.de, okantopcu@gmail.com, robert.siegfried@aditerna.de, oguztuzn@ceng.metu.edu.tr

Keywords: Scenario Development, Distributed Simulation, Base Object Model, Model-Driven Engineering.

Abstract: Scenario development starts with capturing scenarios from the users and leads to the design and the development of the simulation environment to execute these scenarios. This paper proposes a scenario development process adopting a Model-Driven Engineering (MDE) perspective. It takes scenario development and the use of scenarios in simulation environment development put forth in IEEE Recommended Practice for Distributed Simulation Engineering and Execution Process (DSEEP) as a starting point. It then constructs a basic vocabulary including the definitions of operational, conceptual, and executable scenarios. Following MDE principles, scenario development is viewed as a series of model transformations. Operational scenarios, mostly defined in a natural language, are first transformed into conceptual scenarios, which conform to a formal metamodel. Then conceptual scenarios can be transformed into executable scenarios specified using a specific scenario definition language. Furthermore, it is also possible to generate the constructs of simulation environment design and development using model transformations. In this regard, a conceptual scenario metamodel is proposed adopting the Base Object Model metamodel as an example. Then this metamodel is used to present the proposed process with a sample operational scenario and conceptual scenario excerpts. Samples are shown how model transformation can be employed for developing a Federation Object Model and an executable scenario file.

1 INTRODUCTION

Although the importance of scenarios in modelling and simulation has long been well known, there still exists a lack of common understanding and standardized practices in simulation scenario development. Scenario development starts with eliciting scenarios from the users and leads to physical scenario data representation for runtime execution and constraints to simulation environment design.

Scenario development is an extensive process beginning with the stakeholders' descriptions of the scenario and finishing with the generation of the corresponding executable specifications. The scenario development is a part of the simulation environment development process. The scenario development aims at developing a specification of a simulation run, but it is also an input for the design and development of the simulation environment itself.

Siegfried and his colleagues propose to distinguish three types of scenarios that are produced in successive stages of the scenario development process: operational scenarios, conceptual scenarios and executable scenarios (Siegfried et al., 2012) (Siegfried et al., 2013) (MSG-086, 2014).

In this paper Siegfried's definitions are used as a baseline for devising a model-driven scenario development process. This process involves establishing a scenario development pipeline. It adopts Model-Driven Engineering (MDE). MDE proposes that one shall construct a model of the system to be built and then proceed with a series of transformations to obtain an executable system (Mellor et al., 2003). Following the principles of MDE, scenario development is viewed as the transformation of operational scenarios (defined in a natural language) to conceptual scenarios (conforming to a formal metamodel) then to executable scenarios (specified using a specific scenario definition language) and simulation

SIMULTECH 2014 - 4th International Conference on Simulation and Modeling Methodologies, Technologies and Applications

environment design (defined in a particular formalism).

After introducing the required background information, the proposed model-driven scenario development process is presented. Then the process is illustrated with a simple example.

2 BACKGROUND

The definition of scenario has long been a subject of discussion. Siegfried et al. (Siegfried et al., 2012) define a scenario as a specification of conditions and situations to be represented by a simulation environment for its purpose.

In IEEE 1278 (IEEE, 1993), the standard for Distributed Interactive Simulation, a scenario is defined as the description of initial conditions and timeline of significant events. The definition given in the High Level Architecture Glossary (US Department of Defense, 1996) stresses that a scenario shall identify the major entities with their capabilities, behavior and interactions over time with all related environmental conditions. The NATO Science and Technology Organization Modeling and Simulation Group 053 (MSG-053, 2010) defines a scenario as a description of the hypothetical or real area, environment, means, objectives and events during a specified time frame related to events of interest.

The operational scenarios are provided in the early stages of a simulation environment development process by the user or the sponsor. The operational scenarios can be documented in any textual or graphical form. The key elements in a scenario are the initial state, the desired end state, the course of actions to reach the prescribed end state, and the entities with their capabilities and relations.

The operational scenarios provide a coarse description of the intended situation and its dynamics, but they need to be refined and augmented with additional information pertaining to simulation. This refinement is usually done by M&S experts and results in conceptual scenarios. Conceptual scenarios provide a detailed specification of the piece of the world to be represented in the simulation environment and should provide crucial information for everyone who is involved in the later stages of the simulation environment engineering process.

Once a simulation environment is designed and set up, the executable scenarios have to be available for all simulation systems and other member applications of the simulation environment. For this purpose, the conceptual scenarios need to be transformed into executable scenarios. An executable scenario is the specification of a specific situation providing all information necessary for the preparation, initialization, and execution of a simulation environment and for supporting scenario management activities such as scenario distribution and role casting (Topçu & Oğuztüzün, 2010). The transformation from conceptual scenarios to executable scenarios is undertaken primarily by the operator of the member applications of the simulation environment (possibly assisted by M&S experts or subject matter experts). Ideally, the resulting executable scenarios are specified in a way that they are directly processable by the member applications (e.g. as a file containing parameters or via a web service).

3 DEVELOPMENT PROCESS

A standard perspective for the utilisation of scenarios in simulation development and execution is introduced in IEEE Recommended Practice for Distributed Simulation Engineering and Execution Process (DSEEP) (IEEE, 2010a). DSEEP describes a process framework for development and execution of distributed simulation environments. The DSEEP recommends scenario development activity as a part of the problem conceptual analysis. The outcome of this activity is defined as the major entities that must be represented in the simulation environment, description of their capabilities, behaviors and relationships, event timelines, the environment, and the initial and terminal conditions. The DSEEP then prescribes the utilization of scenarios: a) for the design of a simulation environment and for the design of the member applications, and b) for designing and establishing the simulation environment agreements in simulation environment development. These agreements enable the simulation applications to interoperate. From an HLA perspective, this corresponds to defining federates, a Federation Object Model (IEEE, 2010b), and Federation Agreements (Johns Hopkins University Applied Physics Laboratory, 2010).

MDE has also been employed in systems development in the simulation domain to generate elements of a simulation system or simulation environment from models via model transformations (Topçu et al., 2008) (Adak et al., 2009) (Gaševic et al., 2009) (Durak et al., 2009) (Tolk, 2002) (Cetinkaya et al., 2011). The MDE methodology is

regarded as a natural continuation of the advances in raising abstraction level for systems development to boost productivity as well as quality (Atkinson & Kuhne, 2003). The models are refined and transformed during the development process until an executable artifact is obtained.

Adopting this MDE definition for scenario development, the authors propose a development process in which conceptual scenarios are specified based on a metamodel and then executable scenarios for various target simulation systems are generated via model transformations employing transformation rules specified for those particular targets.

Conforming to the process model recommended by DSEEP, we promote the construction of the conceptual scenarios as models and the utilization of model transformations for designing member applications, environment agreements and executable scenarios.

4 METAMODELING

MDE worldview is founded on models and transformations among them. In order to describe a model, one needs a language. One way to provide a language is metamodeling. The Object Management Group (OMG) introduced a four-level metamodeling architecture, which specifies four levels: Information (M0), Model (M1), Metamodel (M2) and Meta-metamodel (M3), and their relations (OMG, 2011b).

M0 consists of the data to be described. M1 comprises the model that describes the data. M2 describes the structure and the semantics of the model and named as metamodel and M3 is the top level that specifies the structure and the semantics of the metamodel and named as meta-meta model.

The proposed model-driven scenario development process is structured upon this four-layer metamodeling architecture of OMG. The process advocates constructing a Conceptual Scenario Metamodel prior to developing conceptual scenarios. The aim is to start with a metamodel to enable building a conceptual model and then to support the model transformations from the source, conceptual scenario to target simulation application design, simulation environment agreements, and executable scenario.

Figure 1 exemplifies the proposed process. It recommends to develop a completely new metamodel or to use an existing one for the shown targets depending on the simulation environment development process. The representations of target models depend on the specification languages used

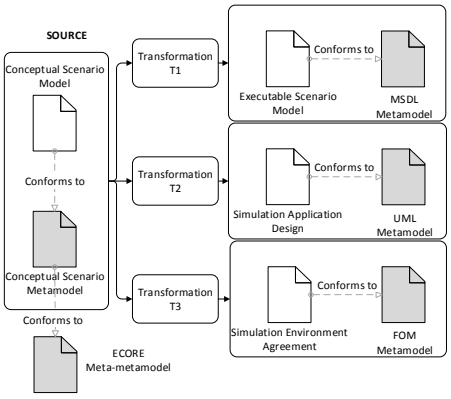


Figure 1 Model-Driven Scenario Development Process.

as depicted in the exemplified transformations in Figure 1. For instance, the design of a simulation application can be specified by using a general modeling language such as UML, or by using a more specific representation which targets a specific platform such as Federation Architecture Metamodel (FAMM) for HLA federations (Topçu et al., 2008). If the UML metamodel is the target the transformation rules will be specified from the Conceptual Scenario Metamodel to the UML metamodel. If HLA Object Models for environment agreement are used, then FOM metamodel is the target. Lastly, one can use specific scenario definition languages such as the Military Scenario Definition Language (MSDL) (SISO, 2008) as the target metamodel for executable scenarios.

In the proposed scenario development process, the conceptual scenario is subject to either model-to-model or model-to-text transformations. To accomplish these transformations, one needs to specify the mappings between the constructs of the source metamodel and those of the target metamodel. Then a source model is transformed into a target model by executing the specified transformation (Gronback, 2009).

5 SAMPLE IMPLEMENTATION

In this section, the process introduced in the previous section will be elaborated using a sample implementation. In this implementation, we will first introduce metamodeling over a conceptual scenario metamodel that has been constructed adopting Base Object Model (BOM) (SISO, 2006) metamodel.

Eclipse Modeling Framework (EMF) is

employed to realize the four-level metamodeling architecture. EMF is defined as a framework for describing models and then generating other constructs, such as other models, code or text from them (Steinberg et al., 2008).

Next, a sample conceptual scenario vignette will be introduced using the conceptual scenario metamodel. Finally, transformation definitions will be discussed over the sample mappings for generating FOM and the executable scenarios from the conceptual scenario.

5.1 Conceptual Scenario Metamodel

Base Object Model (BOM) introduces the interplay, the sequence of events between simulation elements, as well as the reusable pattern, and provides a standard to capture the interactions (SISO, 2006). Siegfried and his colleagues presented BOMs as a method for capturing conceptual scenarios (Siegfried et al., 2013). Following this approach, we adopt the BOM metamodel specified in the standard to construct a conceptual scenario metamodel.

Ecore is provided as the meta-metamodel used to describe metamodels in EMF (Steinberg et al., 2008). To define a metamodel, one makes use of four Ecore classes, namely, EClass, EAttribute, EReference and EDataType. EClass is defined as the modelled class with attributes and references. EAttribute is the modelled attribute with a name and a type. EReference is specified as an association between classes. EDataType is the type of an attribute (Steinberg et al., 2008).

At the top level (Figure 2) *ConceptualScenario,* defined as an *EClass,* has associations that are defined by *EReference* constructs: *entities, stateMachines, interplays, events* and *identification*. These relate *ConceptualScenario* to other *EClasses Conceptual Entity, StateMachine, PatternOf Interplay, Event* and *ScenarioIdentification,* respectively.

ScenarioIdentification has attributes that are defined by *POCEmail, POCTelephone* and so on as *EAttributes*. As an example the *Purpose* attribute is defined as a string *(EString)* data type *(EDataType).*

A conceptual scenario is defined with one or more state machines. A state machine is defined by a number of states. Each state has an exit condition and a next state. Exit conditions are associated with exit actions that are pattern actions. For example, in a flight simulation, the aircraft conceptual entity may have six states: taxi, takeoff, climb, cruise, descend and landing. The exit condition for taxi can be defined as the takeoff clearance given by the control tower. Then takeoff can be specified as the next state.

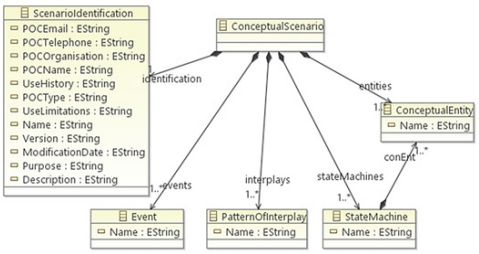


Figure 2: Conceptual Scenario Metamodel Top Level Diagram.

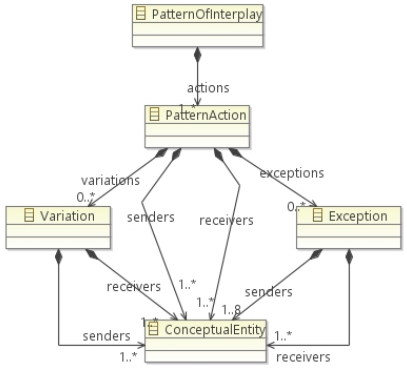


Figure 3: Pattern of Interplay in Conceptual Scenarios.

Patterns of interplay are defined as building blocks in the BOM specification (SISO, 2006). They capture the pattern actions as well as their exceptions and variations. As shown in Figure 3, actions are initiated by sender conceptual entities and receivers are the intended recipients. Exceptions are defined as the actions that cause the remaining sequence to fail. Variations, however, are defined as alternative ways of an action that do not affect the completion or success. Considering again the case of an aircraft, the pattern of interplay for the departure is likely to include the aircraft beginning to roll from its parking position in the direction of the assigned runway. Depending of the parking position (in front of the Terminal or further away on the Apron) an initial push-back might be required or not, which can be introduced as a variation of this action. The sender for the taxiing can be specified as the pilot and the receiver as the aircraft. The second action can then be defined as getting the clearance from the control tower. The sender in this case is the control tower and receiver is the pilot and the event is the takeoff clearance. And the pattern of interplay can

continue with an action of applying power to the aircraft's engine.

In BOM metamodel, conceptual entities take part in patterns of interplay as senders and receivers and each entity is associated with a state machine. Entities possess characteristics and the BOM metamodel is enhanced by adding values to these characteristics to define scenario parameters. For example, various characteristics can be specified for an aircraft entity in a flight simulation scenario, some of which are initial position, fuel weight or gross weight. The values of these characteristics then determine the scenario parameters.

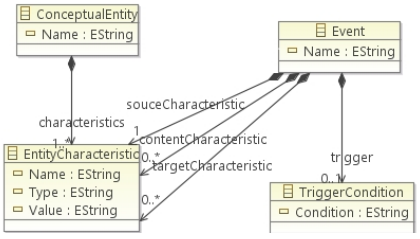


Figure 4: Conceptual Entities and Events in Conceptual Scenarios.

Events (Figure 4) are used to capture the messages and triggers. Triggers present undirected events when a change in the characteristic of an entity creates a response from other entities. The condition of change is captured in a trigger condition. Messages are directed events from one entity to another that are uniquely identified by the source and target characteristic. The content of a message is given in content characteristics. As an example, takeoff clearance is a message from tower (which is identified by its airport id) to an aircraft (which is identified by its call sign). The content of the message is the takeoff clearance characteristic of the aircraft. When the takeoff characteristic of an aircraft is true, then it is a trigger event. Then the pilot entity can start an action for gears up.

5.2 Model-Driven Conceptual Scenario Development

This section is based on an operational scenario for the departure activity of an aircraft. Below is an extract from the operational scenario.

"Aircraft D-ATRA stands in front of its hangar at DLR in Braunschweig. Pilots ask the tower for taxi clearance. The tower provides taxi instructions towards RWY 08. Pilots then start taxiing according to instructions. Then tower provides information about the departure like weather, VRB05KT, R08/2800FT, overcast sky. Then pilots ask for a departure clearance and tower grants the departure." (DFS Deutsche Flugsicherung GmbH, 2013) (Ahmad & Sexana, 2008).

This is an example operational scenario one can obtain from the users or sponsors of a flight simulator. It is obvious that not all data that is required to run this scenario is available in this text. An M&S expert needs to augment the missing information and to develop a conceptual scenario.

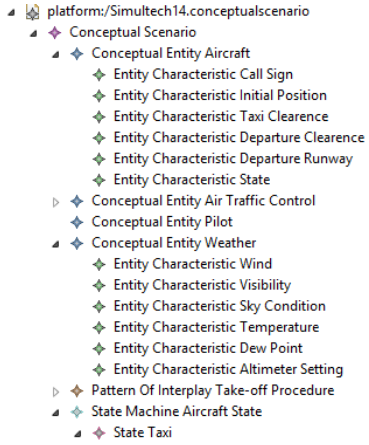


Figure 5: Conceptual Scenario Editor Tree Viewer.

EMF.Edit (Steinberg et al., 2008) is employed to build conceptual scenario editor via automatic code generation to display and edit the instances of the developed metamodels. This editor provides a tree viewer and properties sheet for each conceptual scenario element. Figure 5 presents the tree viewer for the sample conceptual scenario. The tree presents some of the conceptual entities from the operational scenario such as aircraft, pilot, and weather. The main pattern of interplay is defined as takeoff procedure. Aircraft state is captured as a statechart. When developing the conceptual scenario, missing weather characteristics in operational scenario such as temperature and dew point are also added to the model.

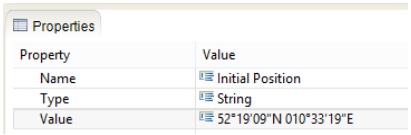


Figure 6: Conceptual Scenario Editor Properties Viewer.

SIMULTECH 2014 - 4th International Conference on Simulation and Modeling Methodologies, Technologies and Applications

The Properties viewer (Figure 6) enables a user to specify the attributes of the model elements. As an example, the attributes of initial position entity characteristics are its name (initial position), type (string), and value (52°19′09″N 010°33′19″E). Thus an M&S expert can specify the implicit reference to the initial location of the aircraft in the operational scenario explicitly.

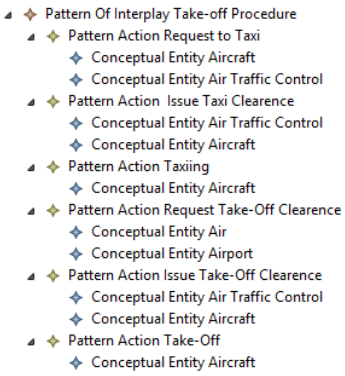


Figure 7: A Sample Pattern of Interplay.

The takeoff procedure is presented in Figure 7 as a sample pattern of interplay. There are six consecutive actions, starting from “aircraft requests to taxi from Air Traffic Control” till its takeoff. The sender and receiver entities are all captured. Even though exceptions and variations can be specified, this sample does not exhibit any.

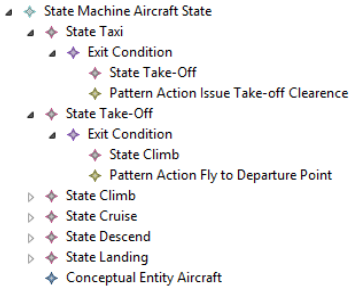


Figure 8: A Sample State Machine.

Figure 8 introduces a sample aircraft state machine. Aircraft states include taxi, takeoff, climb, cruise, descend and landing. The next state after taxi is takeoff and the exit action of the taxi state is the issue of takeoff clearance. The next state after takeoff is climb and takeoff ends with the action flying to departure point.

5.3 Model Transformations

Model transformations are the enabling tools of MDE for development. Throughout the engineering process, models are the main artifacts, and transformations enable the reflection of the information captured in one model to another one as well as enriching the source model with specialized information. In the model-driven scenario development process model transformations are proposed for transforming the information that is captured in a conceptual model to simulation environment design, simulation environment agreements and executable scenarios. To define transformations from a source metamodel to a target metamodel, a model transformation language is required. Atlas Transformation Language (ATL) (Jouault et al., 2006), Graph Rewriting and Transformation (GReAT) (Agrawal, 2003) and Query / View / Transformation (QVT) (OMG, 2011a) are some of the commonly used languages. Rather than addressing any specific model transformation language, users are recommended to pick any model transformation language that fits their specific requirements such as the development environment requirements or target model requirements.

Here, the sample transformation specification, or mapping, was developed using QVT utilizing Eclipse Model-to-Model Transformation (MMT) project (The Eclipse Foundation, 2014). It supports a QVT Operational, a partial implementation of the QVT specification (Barendrecht, 2010).

```
-- model type definition to conceptual --
scenario metamodel
modeltype CS uses 'ConScen.ecore';
-- model type definition to UML
modeltype UML uses 'SimpleUML.ecore';
-- transformation definition from
-- Conceptual Scenario to UML
transformation scenario2UML(in CS :
                  ConSce, out UML);
-- main triggers the transformation
main(in scenario: CS::ConSce,
                  out umlModel: UML::Model)
{
   umlModel := scenario.map scen2UML ();
}
```

As presented above, QVT defines transformations using a certain structure, which consists of model type definitions, transformation declarations, and a main function. Metamodels are referred by using model type definitions. Transformation declarations specify the input and the output metamodels. The main function starts the transformation process by calling the first transformation.

Mappings specify which object from an instance of a source metamodel will be transformed to which specific object in the instance of the target metamodel. The declarations identify the source class name and the target class name. One can also specify constraints to mappings using Object Constraint Language (OCL) (OMG, 2006). In the body of a mapping, the variables and the parameters are initialized in the `init` section, mappings are specified in the `population` section and post-processing can be done in the `end` section.

```
mapping CS::CS:: scen2UML() : UML::Model
{
   init { log("Mapping Started!"); }
   packages   := self.entities2packages();
   interfaces := self.entChar2intClasses();
   states     := self.states2staClasses();
    ...
   end { log("Mapping Ended!"); }
}
```

Above is a representative excerpt from the top level mapping that is called in the main function. Conceptual entities in the conceptual scenario metamodel are mapped to packages in the UML model calling a new mapping function *self.entities2packages()*. Similarly, the entity characteristics and the states in the conceptual scenario metamodel are mapped to the interface classes and to the state classes, respectively, in the UML model.

```
mapping CS::CS:: scen2FOM() : FOM::Model
{
   ...
   objects      := self.entities2objects();
   objectAttr   := self.entChar2objAttr();
   interactions := self.events2intact();
   intParam     := self.contChar2intParam();
   ...
}
```

Likewise, a sample portion is provided for the conceptual scenario to Federation Object Model transformation. In this case entities can be mapped to HLA objects and entity characteristics to object attributes. Events in the conceptual scenario metamodel can be mapped to HLA interactions in a FOM and content characteristics to interaction parameters.

```
mapping CS::CS:: scen2EXE() : EXE:File
{
   ...
   entitites    := self.entities2entities();
   initialCond  := self.entChar2iniCond();
   injectEvents := self.events2inject();
   logData      := self.states2logging();
   ...
}
```

For a transformation to create an executable scenario from a conceptual scenario, mappings need to be specified as exemplified above. Entities in the conceptual scenario must be mapped to the entities of the executable scenario.

6 CONCLUSIONS

This paper introduced a model-driven scenario development process which is based on the explicit specification of conceptual scenarios using a metamodel. The proposed development process recommends the use of model transformations to generate the executable scenarios. Practitioners of this process shall develop their metamodels for the source (i.e. conceptual scenario) and target (i.e. executable scenario, design of simulation applications, or simulation environment agreements).

The proposed process is illustrated with a simplified case study. In this respect, BOM presents a prospect in specifying the conceptual scenarios as the source metamodel. Target metamodels on the other hand are more or less application-specific. The examples introduced in this paper made use of Eclipse-based technologies for modeling and transformation,although there exist alternatives to Eclipse for each of these steps.

In order to elaborate on the proposed model-driven scenario development process, an effort to develop the corresponding workflows will be worthwhile.

REFERENCES

Adak, M., Topçu, O. & Oğuztüzün, H., 2009. Model-based Code Generation for HLA Federates. *Software: Practice and Experience*, 40(2), pp.149-75.

Agrawal, A., 2003. *GReAT: A Metamodel Based Model Transformation Language*. Institute for Software Integrated Systems (ISIS), Vanderbilt University.

Ahmad, S. & Sexana, V., 2008. Design of Formal Air Trafic Control System throug UML. *Ubiquitous Computing and Communication Journal*, 3(6), pp.11-20.

Atkinson, C. & Kuhne, T., 2003. Model-driven development: A metamodeling foundation. *IEEE Software*, 20(5), pp.36-41.

Barendrecht, P.J., 2010. *Modeling transformations using QVT Operational Mappings*. Research project report. Eindhoven: Eindhoven University of Technology Department of Mechanical Engineering Systems Engineering Group.

Cetinkaya, D., Verbraeck, A. & Seck, M.D., 2011. MDD4MS: A Model Driven Development Framework

for Modeling and Simulation. In *Summer Simulation Conference*. The Hague, Netherlands, 2011.

DFS Deutsche Flugsicherung GmbH, 2013. *Aerodome Chart -ICAO Braunschweig-Wolfsbug*. Langen: DFS Deutsche Flugsicherung GmbH.

Durak, U., Oguztuzun, H. & Ider, K., 2009. Ontology Based Domain Engineering for Trajectory Simulation Reuse. *International Journal of Software Engineering and Knowledge Engineering*, 9(8), pp.1109-29.

Gaševic, D., Djuric, D. & Devedžic, V., 2009. *Model Driven Engineering. In Model Driven Engineering and Ontology Development*. Berlin: Springer.

Gronback, R.C., 2009. *Eclipse Modeling Project: A Domain-Specific Language*. Upper Saddle River, NJ: Addison-Wesley.

IEEE, 1993. IEEE 1278 *Protocols for Distributed Interactive Simulation Applications-Entity Information and Interaction*. Standard. New York, NY: IEEE.

IEEE, 2010a. *IEEE Recommended Practice for Distributed Simulation Engineering and Execution Process (DSEEP)*. New York, NY: IEEE.

IEEE, 2010b. *IEEE Standard for Modeling and Simulation High Level Architecture (HLA)– Object Model Template (OMT) Specification*. Standard. New York, NY: IEEE.

Johns Hopkins University Applied Physics Laboratory, 2010. *Live-Virtual-Constructive Architecture Roadmap Implementation, Common Capabilities—Federation Agreements Template Users' Guide*. Technical Report. Laurel, MD: Johns Hopkins University.

Jouault, F. et al., 2006. ATL: a QVT-like Transformation Language. In *OOPSLA '06*. New York, 2006. ACM.

Mellor, S.J., Clark, A.N. & Futagami, T., 2003. Guest editors' introduction: Model-driven development. *IEEE Software*, 20(5), pp.14-18.

MSG-053, 2010. *Rapid Scenario Generation for Simulation Applications*. RTO Technical Report. Brussels: NATO.

MSG-086, 2014. *Guideline on Scenario Development for (Distributed) Simulation Environments*. STO Technical Report. Brussels: NATO.

OMG, 2006. *Object Constraint Language, Version 2.0*. Standard. Needham, MA: OMC Object Management Group.

OMG, 2011a. *Meta Object Facility (MOF) 2.0 Query/View/Transformation Specification*. Standard. Needham, MA: OMG.

OMG, 2011b. *Object Management Group, Meta object facility,MOF specification version 2.4.1*. Standard. Needham, MA: OMG.

Siegfried, R. et al., 2012. Scenarios in military (distributed) simulation environments. In *Spring Simulation Interoperability Workshop (S-SIW)*. Orlando, 2012. SISO.

Siegfried, R. et al., 2013. Specification and documentation of conceptual scenarios using Base Object Models (BOMs). In *Spring Simulation Interoperability Workshop*. San Diago, 2013. SISO.

SISO, 2006. SISO-STD-003-2006 *Base Object Model (BOM) Template Specification*. Orlando, FL : Simulation Interoperability Standards Organization (SISO).

SISO, 2008. *Standard for Military Scenario Definition Language (MSDL)*. Standard. Orlando, FL : SISO.

Steinberg, D., Budinsky, F., Merks, E. & Paternostro, M., 2008. *EMF: Eclipse Modeling Framework*. 2nd ed. Upper Saddle River, NJ: Pearson Education.

The Eclipse Foundation, 2014. *Model-to-Model Transformation (MMT)*. [Online] Available at: https://projects.eclipse.org/projects/modeling.mmt [Accessed 10 February 2014].

Tolk, A., 2002. Avoiding another Green Elephant – A Proposal for the Next Generation HLA based on the Model Driven Architecture. In *Fall Simulation Interoperability Workshop*. Orlando, FL, 2002.

Topçu, O., Adak, M. & Oğuztüzün, H., 2008. A Metamodel for Federation Architectures. *Transactions on Modeling and Computer Simulation (TOMACS)*, 18(3), pp.10:1-10:29.

Topçu, O. & Oğuztüzün, H., 2010. Scenario Management Practices in HLA-based Distributed Simulation. *Journal of Naval Science and Engineering*, 6(2), pp.1-33.

US Department of Defense, 1996. *High Level Architecture Glossary*. US DoD: Washington DC, VA.

B.3 User-Guided Transformations for Ontology Based Simulation Design

Ozer Ozdikis, Umut Durak and Halit Oguztuzun. 2009. User-guided transformations for ontology based simulation design. In *Proceedings of the 2009 Summer Computer Simulation Conference*, SCS, Istanbul, Turkey, 75-82.

User-Guided Transformations for Ontology Based Simulation Design

Özer ÖZDİKİŞ, Siemens EC, ozer.ozdikis@siemens-enterprise.com
Umut DURAK, TUBİTAK-SAGE, udurak@sage.tubitak.gov.tr
Halit OĞUZTÜZÜN, Middle East Technical University, oguztuzn@ceng.metu.edu.tr

Keywords: Ontology based simulation, domain engineering, ontology languages, object-oriented design methods, model driven simulation development.

Abstract

Using domain knowledge represented as ontologies for the design of domain specific architectures is a promising approach for better reusability. Tool support is essential for this approach to be effective. We present a flexible, user-guided transformation method to generate a framework architecture model from a domain model. The latter is in the form of an OWL ontology, and the former is in the form of a UML class diagram. We introduce a transformation tool that allows the user to configure mappings from the source OWL constructs of the simulation ontology to the target UML constructs of the simulation design. This user-guided approach is demonstrated on a case study that involves building a framework architecture for ontology-driven trajectory simulation development.

1. INTRODUCTION

Transformation of domain models, in the form of OWL ontologies, to framework architectures, in the form of UML class diagrams, is an issue that arises in the context where domain engineering and model-driven development meet. We hold the view that this issue can be resolved to the satisfaction of the user with the aid of the user-guided model transformation tools.

The use of models to represent the outcomes of domain engineering activities, and the use of model transformations and code generation as a means to carry out these activities bring Model Driven Development (MDD), in particular, OMG's Model Driven Architecture (MDA), to bear on Domain Engineering [1], [2].

As information is gathered during domain analysis, representing the knowledge for ease of both human understanding and machine processability becomes a problem [3]. Using ontologies is proposed as a solution to this problem [4]. Using ontologies as domain models is called ontology based domain engineering. One of the promises of ontology based domain engineering is to derive reusable components from ontologies [5]. Referring to the potential benefits of ontologies for modeling and simulation that are listed by Miller and Fishwick in [6], in ontology based domain engineering applied to simulation development, ontologies are used as domain models that enable simulation reuse at knowledge, design and code level.

Ontology as a domain model, however, is the subject matter expert's view of the domain. Tool-supported methods of generating the software developer's view of the domain is desirable. Our present focus is on tool support for flexible transformations from a domain ontology into a UML class diagram.

As an ontology language, W3C's OWL is a well accepted semantic Web standard [7], [8]. When it comes to transforming available OWL ontologies into UML class diagrams, the "one size fits all" approach does not work. Because every domain model may require a different transformation procedure depending on the context, data types, conventions, and even the personal preferences of the software engineer who specifies the reuse infrastructure. A hard-coded transformation approach would not be flexible enough in general to be effective in practice. What we propose is a configurable transformation approach: Mappings from OWL ontology constructs to UML class diagram constructs will be guided by the user.

Our implementation for this approach provides a user interface to design mappings between the OWL and UML metamodel constructs. These mapping definitions are applied on the OWL ontology (domain model), and a UML class diagram (as a constituent of a framework architecture) is produced. This reconfigurable transformation tool has been employed in an industrial development effort, in which a domain ontology, called Trajectory Simulation ONTology, in short TSONT [9], is transformed to an object oriented application framework architecture. This case study will be discussed at length.

2. BACKGROUND

Three levels of abstraction are identified in OMG's Model Driven Architecture (MDA) for the modeling of systems [1], [2]. These are Computation Independent Model (CIM), Platform Independent Model (PIM), and Platform Specific Model (PSM). CIM is a view of a system from the computation independent viewpoint. PIM is a view of a system from the platform independent viewpoint. PIM describes the system, but does not refer to the platform on

which the system is implemented. PSM, in contrast, refines the PIM with the details that specify how that system utilizes a particular platform. PSM is supposed to be readily transformable to executable code.

The idea is to capture domain knowledge at a conceptual level with CIM and generate the models at PIM and PSM as automated as possible, to finally get a running code. Ontologies can serve as CIMs. They are the representation of the domain information prepared by the domain and knowledge engineers with the help of subject matter experts, without any concern about system development. Our focus is to allow the developer to guide the transformation from a CIM, which is represented by an OWL ontology, to a PIM, which is represented by a framework architecture description involving UML class diagrams.

Our transformation method from CIM to PIM (i.e. from OWL to UML) can be located in reference to the four-layer metamodeling hierarchy of OMG's Meta Object Facility (MOF) [10]. MOF is an extensible model driven integration framework for defining, manipulating, and integrating metadata and data in a platform independent manner. It provides a meta-metamodel at the top level, which is called M3 layer of the four-layer metamodeling hierarchy. MOF can be used to define more specific metamodels at M2 layer, such as the UML metamodel. We have defined a simplified simplified UML metamodel complying with the OMG's specification [11]. UML models conforming to the UML metamodel are at layer M1. Finally, the M0 layer includes the instances created from a UML model.

Similar to UML models, ontology models must also conform to a metamodel. Our ontology modeling hierarchy is based on Eclipse Metamodel Framework (EMF) [12]. The meta-metamodel at M3 layer in EMF is called Ecore. IBM implemented an Integrated Ontology Development Toolkit (IODT) for ontology driven development built on EMF [13]. IODT includes a library called EMF Ontology Definition Metamodel (EODM) which is an implementation of the OMG's Ontology Definition Metamodel [14]. EODM has OWL parsing, serialization, and reasoning features.

In the transformation from CIM to PIM, we have an ontology model as our source and a UML model as our target. Our approach facilitates the definition of the mapping rules between the EODM and UML metamodels at M2 layer. These rules are applied to a given OWL ontology to generate a UML class diagram. The modeling layers which are used in this transformation are shown in Fig. 1.

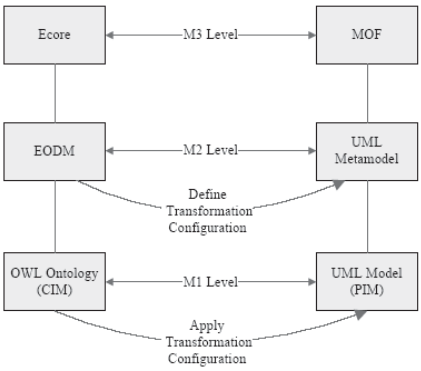


Fig. 1 - Relations between the modeling layers

3. THE OWL-TO-UML TRANSFORMATION TOOL

Our tool lets the user configure the transformation he needs. The transformation configuration is composed of transformation rules, mappings and constraints.

A rule is a collection of mappings from a specific OWL construct to some UML constructs. In other words, a rule includes mappings and a definition of source OWL constructs to apply these mappings. Source constructs can be OWL classes or OWL properties.

Following the description of the source construct, the user must specify, in terms of mappings, how to use them in the transformation. Depending on the source-target combinations, different mapping possibilities are provided to the user. The user can define three types of mappings in a rule: to a UML class, to an association, and to an operation parameter. A mapping is the prescription of how the target UML construct should be built using the source OWL construct.

Constraints can be defined to restrict the entities to be evaluated in a specific rule or mapping. Constraints are actually condition-value pairs applied on an OWL construct. While evaluating an OWL construct in a rule or mapping, these condition-value pairs are used to find out if that construct is selected and should be processed.

3.1. Available OWL Constructs at the Source

An OWL ontology involves classes, object properties, and datatype properties. A class has a classname and possibly super classes. A class may be defined as an intersection of, union of or complement of other classes. Further, a class may have different types of restrictions. These restrictions define the values (restrictionValue) that the class must take for a property (onPropertyName). A class may also be an enumeration class, which includes the list of individuals that are the members of the class. A property has a property name, domain and range information, and possibly super properties. Our tool lets the

user use definitions of classes, object properties, and datatype properties as a source for the mappings to UML.

3.2. Available UML Constructs at the Target

A UML model can include the definitions for the UML classes, generalizations, associations, attributes, operations, and parameters for operations. A class has a name and possibly some attributes and operations, which are identified by their names within a class. An operation can have parameters of type "in", "out", "in-out", and "return". There can be generalizations and associations defined between classes. Generalizations represent the subclass-superclass relationships. Associations can be of type aggregation, composition, or directed association, possibly with multiplicity information attached at either end. One restriction in this metamodel definition is that the class names in the model and operation names in a class must be unique. This is because the OWL classes in the ontology are also identified with their names.

3.3. Mappings from source OWL to target UML

Our tool provides an interface to the user to define mappings between the available OWL constructs at the source and the UML constructs at the target, which were defined above. Mappings can be classified into 3 types regarding the target constructs. These are

- Mappings to UML classes
- Mappings to UML associations
- Mappings to UML operation parameters

During the transformation process, mappings to UML classes are resolved first. After the classes are ready in the target UML model, mappings to associations, and finally, operation parameters are evaluated. The name of the ontology is carried over to the target model as the package name.

3.3.1. Mappings to UML Classes

This mapping type is used to introduce new classes into the UML model, with their super class relationships, attributes, and operation names. Depending on the type of the rule, OWL classes or OWL properties in the input ontology are traversed considering the constraint definitions. For each selected OWL class/property, a new UML class is created in the UML model with the same OWL class/property name. If a class with the same name already exists in the UML model, it is not created again since the class names are supposed to be unique in a UML model according to our metamodel definition. In addition to the new UML class definitions, user can define further mappings between the OWL and UML class constructs. The available OWL constructs can be used to define the super classes, attributes, or operation names of the corresponding UML class.

3.3.2. Mappings to UML Associations

This mapping provides the opportunity to define associations between the UML classes in the target UML model. OWL classes/properties in the input ontology are traversed according to the constraint defined for the rule. The name of the currently traversed OWL class/property is used to identify the first end point in the UML association. Second end point in the association is found depending on the rule type and the mapping configuration. In the mapping definition, the user selects the association type to be used, where the choices are aggregation, composition, and directed association.

In the mapping, in addition to the association type, it is also possible to define constraints for each unique association definition. A special case for the min/max cardinality restrictions is that the user can define multiplicity of the association. The min/max cardinality restrictions have integer values. If the second association end is to be taken from the onPropertyName of the restriction, value of the restriction can be attached to the association as a multiplicity. This value can be used as exact, lower bound or upper bound multiplicity. End1 or end2 can also be selected for the desired placement of the multiplicity.

3.3.3. Mappings to UML Operation Parameters

There may be cases where some class in the ontology defines an operation with parameters. This mapping is used to fill the parameter names of operations in UML classes. The transformation for this type of mapping works as follows: OWL classes/properties in the input ontology are traversed one by one according to the constraint definitions for the rule. For each selected OWL class/property named C, an operation with name C is searched in UML classes of the target model. For each operation named C, the OWL construct defined in the mapping is used to feed the parameter names to this operation. This mapping type is evaluated as the last mapping type, so that we already have the operation names available in the UML classes of the result model.

The precondition for this mapping is that the correct operations have already been defined in desired UML classes. This is achieved by the operation name mappings in UML class transformations. Thus, mapping to UML operation parameters is just a matter of feeding the parameter names to the previously defined operations of previously defined classes.

The name of the owner UML class can be bound to a constraint in the mapping. In the example scenario, with a constraint defined for the owner class name, hasUnit could

have been filtered out. So the parameter settings for Quantity operation in this UML class would have been ignored. It is also possible to define conditions on each individual parameter name. Parameter types can be defined as one of the four specified in the standard [11], namely "in", "out", "in-out", or "return". Using the constraint definitions on parameter names, user can add different types of parameters to an operation in different mapping configurations.

We provided graph panels to the user for the design of mappings between the OWL and UML constructs. Each graph corresponds to a rule. On these graphs, cells that represent the OWL and UML constructs are tied to each other, which correspond to mappings in a rule. Each graph includes exactly one cell for the OWL construct and one or more cells for the UML constructs.

The graph cell for the OWL construct represents the OWL classes or properties in the source, depending on the rule type. This graph cell enables the user to define constraints on the source OWL constructs.

The type of the UML graph cell defines the target UML construct to which we want to transform the constructs in the source. There are 3 types of UML graph cells corresponding to the 3 different mapping types. What is visualized within the graph is an abstract view of the mapping. User sets the details of the mappings by right clicking on the graph cells and making configurations on separate windows. The whole transformation configuration can be exported to files so that it can be imported and used later again.

Rules, mappings, and constraints can be defined in any order. Once they are defined, user can start the transformation process. The tool traverses all rule definitions, identifies the selected OWL classes/properties according to the constraints and adds UML classes to the UML model. After this step, all UML classes will be available in the UML model to define their operations, attributes, superclass relationships, and associations. This is achieved by a second traversal of the mapping graphs. After all the mappings are processed, all UML structures in the target model are serialized into an XMI file [15]. This XMI file can be imported and interpreted by any standard-compliant UML tool.

4. CASE STUDY: FROM TSONT TO SIMULATION FRAMEWORK ARCHITECTURE

The tool is applied to build an ontology based trajectory simulation framework [16]. The trajectory simulation ontology called TSONT [9] serves as a domain model for the trajectory simulation development projects at TUBITAK-SAGE. TSONT is a domain model, where the domain is trajectory simulation, rather than, say, aerodynamics. Note that the mathematical models in a computable form and the methods of computing them are included in TSONT. Thus, TSONT essentially serves as a simulation conceptual model, in the sense of Pace [17]. Moving from such a model to an architecture model is in parallel with moving from an analysis model to a design model, in the software engineering sense, see Pressman [18].

A development methodology is formulated to reuse this ontology based on Model Driven Engineering. In the reuse scenario, the domain model (i.e. TSONT) is transformed to the Platform Independent Framework Architecture as a step of the abstract design process. This transformation is accomplished with our transformation tool presented in this paper. The same transformation was previously carried out by hand.

There are six main classes at the top of the trajectory simulation ontology [9]. The most important for the transformation is the Trajectory_Simulation_Class, because each of its subclasses should be represented as a UML class in the resulting model. Operation and attribute names are adapted from the OWL restriction definitions for each class. Subclasses of Trajectory_Simulation_Function provide the operation parameter information. The hierarchy including some selected example classes in TSONT is shown in Fig. 2.

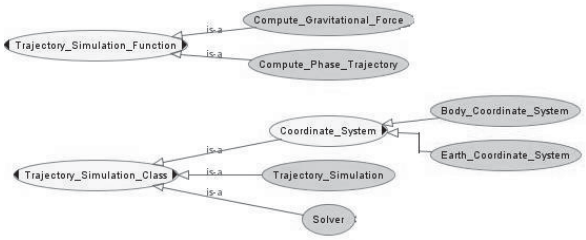


Fig. 2 - Selected Example Classes from TSONT

We define two rules to fulfill the transformation requirements for the trajectory simulation ontology. In the first rule, we define mappings for UML classes and UML associations using the subclasses of Trajectory_Simulation_Class as our source. The second rule is used for mappings to UML operation parameters, essentially, for setting the parameter names in the target UML operations. Since this information is taken from the subclasses of Trajectory_Simulation_Function, we had to define a separate rule to have a separate set of source OWL classes. The first rule can be described as "the rule for classes and associations", whereas second rule as "the rule for operation parameters".

Rule-1: Rule for classes and associations:

The mappings defined in this rule must be applicable only for a restricted set of OWL classes. The criterion to find the source OWL classes is the subClassOf relationship with the Trajectory_Simulation_Class. So we define a constraint in this rule for subclasses, with condition "equals" and constraint value "Trajectory_Simulation_Class". As a result, the source OWL classes are restricted to be the subclasses of Trajectory_Simulation_Class for all mappings in this rule.

This rule will include five mappings. With these mappings, (1) the UML classes will be selected, (2) the subclass-superclass relationships will be defined, (3) operation names in the target UML classes will be selected, (4) attribute names for the target UML classes will be defined and (5) associations between the target UML classes will be created. As a result, for this first rule and its 5 mappings, we design a configuration in our transformation tool. A screenshot from our transformation tool for this rule is shown in Fig. 3. It is represented with the tabbed panel labeled "Rule0".

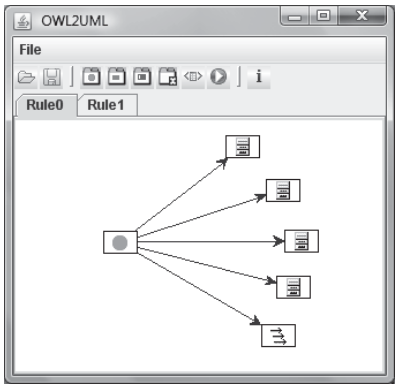


Fig. 3 - Configuration for the first rule

In the above panel, the icon on the left, which is the source of the mapping arrows, represents the source OWL constructs in the source model. Right clicking on this icon opens a window to define constraints for this rule. In this window we define the constraint which is shown in Fig. 4.

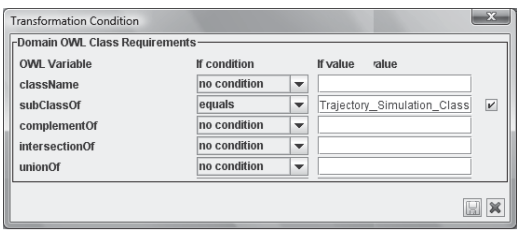


Fig. 4 - Constraint for the first rule

According to the constraint in Fig 4, only the OWL classes which are subclasses of the Trajectory_Simulation_Class will be processed, and others will be ignored.

The first 4 icons on the right side of Fig.3 represent mappings to UML classes, whereas the 5th icon is a mapping to UML associations. User can add any number of mappings in a rule. In this scenario, we added 4 class mappings and 1 association mapping in our first rule. Right clicking on these icons opens new windows to configure each specific mapping.

Mapping-1: Definition of OWL Classes:

All OWL classes which are subclasses of Trajectory_Simulation_Class must be created as a UML class. This is accomplished by adding a UML class mapping. With this mapping, the tool traverses all OWL classes and if an OWL class is found to be the subclass of Trajectory_Simulation_Class, it adds a new UML class to the target UML model. The name of the new UML class is same as the OWL class name. As a result of this definition, we will have the classes like Coordinate_System, Body_Coordinate_System, Earth_Coordinate_System, Solver and all of their subclasses in the resulting UML model.

Mapping-2: Definition of super classes:

One other requirement is that, if two OWL classes have a subclass relationship, this relationship should also be shown in the UML model. This is achieved in the second UML class mapping with a mapping configuration for super classes. The mapping configuration is shown in Fig. 5. With this configuration, for each subClassOf relationship in OWL, a new super class relationship is added in UML between the corresponding sub/super classes. For example, Body_Coordinate_System will be represented as the subclass of Coordinate_System in the resulting UML model.

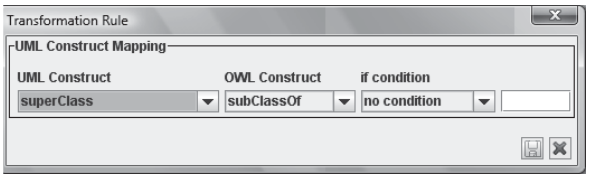


Fig. 5 - Mapping configuration for super classes

Mapping-3: Definition of operation names:

The operation names for each class are to be taken from the AllValuesFrom restrictions if the restriction is defined on a property whose name has the prefix "serves". It is expected that if an OWL class has an AllValuesFrom restriction defined on a property named "servesX", all

values at the restriction should be created as an operation for the corresponding UML class. As an example, the OWL class Trajectory_Simulation has an AllValuesFrom restriction on the property servesComputeTrajectory and the value of this restriction is Compute_Trajectory. So an operation named Compute_Trajectory will be added to the UML class Trajectory_Simulation as a result of this mapping configuration shown in Fig. 6.

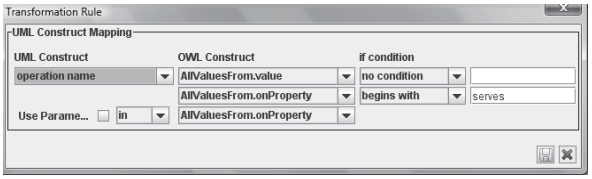


Fig. 6 - Mapping for the definition of operation names

Mapping-4: Definition of attributes:

Attribute information for each class is to be taken from the AllValuesFrom restriction. However in this case, the restriction must be defined on a property whose name has the prefix "has" and the value of the restriction must be used as attribute names of the target UML class. To achieve this, we add a fourth UML class mapping and configure it as in Fig. 7. For example, the OWL class Trajectory_Simulation has an AllValuesFrom restriction on the property hasPhase and the value of this restriction is Trajectory_Simulation_Phase. So an attribute named Trajectory_Simulation_Phase will be added to the UML class Trajectory_Simulation as a result of this mapping configuration.

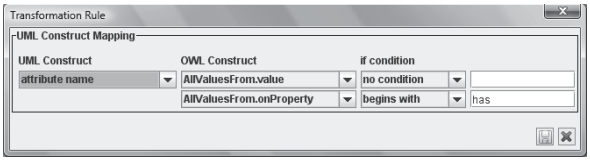


Fig. 7 - Mapping for the definition of attributes

Mapping-5: Definition of associations:

Similar to the attribute definitions, associations must also be taken from the AllValuesFrom restriction values if the restriction is defined on a property whose name has the prefix "has". The 5th icon in the rule panel in Fig.3 represents the association mapping to define aggregation associations. Using the same example as the attributes, we will have an aggregation association from Trajectory_Simulation class to the Trajectory_Simulation_Phase class in the resulting UML model.

Rule-2: Rule for operation parameters:

In Mapping-3 of the first rule, we defined which operations a UML class should have. Now we must define mappings to identify from where to get the parameter names for these operations. The source OWL classes for the parameter names are subclasses of Trajectory_Simulation_Function. So we add a new rule tab in our transformation tool, which is labeled as "Rule1" in Fig.8.

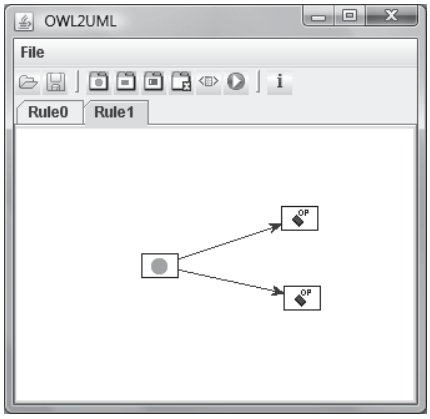


Fig. 8 – Configuration for the second rule

Since we must process only the subclasses of Trajectory_Simulation_Function, we define a constraint in this rule for subclasses, with condition "equals" and constraint value "Trajectory_Simulation_Function" as shown in Fig.9.

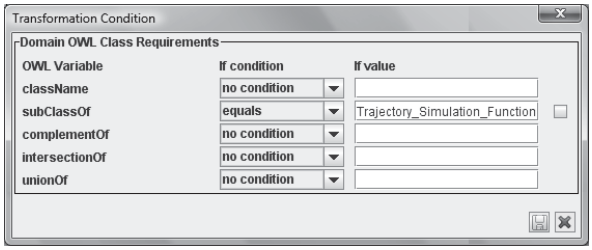


Fig. 9 – Contraint definition for the second rule

Operation parameters are supposed to be taken from the OWL classes with the same name as the operation name in a UML class. These parameter names must be the values of AllValuesFrom restrictions. If the restriction is defined on a property whose name begins with "in", the parameter type should be "in". If the property name begins with "out", parameter type should be "out" (Fig.10). For these "in" and

"out" types, we define two operation parameter mappings in this rule.

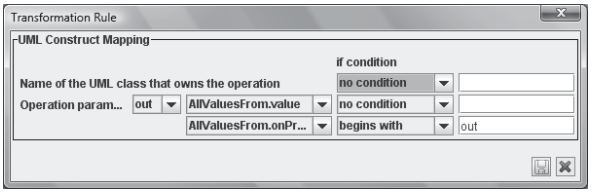


Fig. 10 - Mapping for the definition of parameters type "out"

For example Trajectory_Simulation class has the operation Compute_Trajectory as a result of Mapping-3. During the evaluation of the mappings in this rule, it is found that the Compute_Trajectory has an AllValuesFrom restriction on the property outTrajectory and the value of this restriction is Trajectory. So the parameter Trajectory of type "out" is added to the Compute_Trajectory operation in Trajectory_Simulation class of the target UML model.

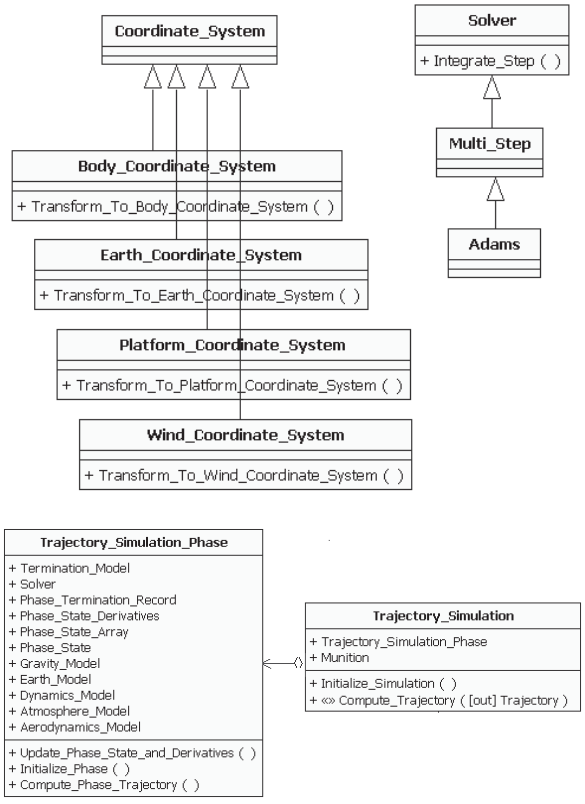


Fig. 11 - Resulting UML class diagram

After the definitions of these rules and mappings, we performed the transformation and the UML class diagram represented as an XMI file is produced, which is successfully imported by Rational Rose XDE [19] and ArgoUML [20]. Fig. 11 is a sample from the resulting UML class diagram. Only the important UML structures are shown in this sample, including the examples above.

We have developed an object-oriented framework based on the resulting class diagram, and constructed several trajectory simulation applications by framework completion. An account of knowledge, design and code reuse thus achieved appears in [16].

5. RELATED WORK

An early effort on transformation of ontologies into UML class diagrams is DUET [21]. It provides a UML based environment for visualizing, developing, and manipulating DAML-OIL ontologies. DUET is a plug-in for Rational Rose 2000 and ArgoUML, which can import DAML-OIL ontologies as UML class diagrams into the UML tool, and export a class diagram as an ontology. The most obvious problem with the approach of DUET is that, it does not provide any flexibility to guide the transformation process. In other words, the rules of transformation are hard-coded and cannot be easily modified depending on the user needs. Moreover, it has a tool dependency since it runs only with Rational Rose 2000 and ArgoUML.

With the latest releases of IODT (since release 1.1.2), EODM includes a transformation feature from OWL models to EODM Ecore models [22]. This enables the interoperability between OWL models and other EMF supported models. This is because the EODM Ecore model can be used as an intermediate model to support transformation between OWL and other modeling languages. Once the OWL is represented as an EODM Ecore model, it can be transformed to a UML Ecore model through MTF [23] transformations. MTF is a rule based transformation framework developed by IBM, which provides a language to define mappings between models conforming to the EMF specifications. However, because of the differences in the expressiveness of OWL and Ecore, this approach does not produce the expected results. Some information is lost during this hard-coded, non-configurable transformation from OWL models to EODM Ecore model, such as the restriction definitions or inter-class relationships. The information represented in the resulting EODM Ecore model is very limited, considering the full set of OWL constructs. Moreover, even after the production of EODM Ecore model, design of MTF rules to generate the UML Ecore model requires in-depth know-how about MTF and Ecore structures.

6. CONCLUSIONS AND FUTURE WORK

The specific contribution our work is a tested tool for ontology based simulation design by introducing a user-guided transformation process to bridge the gap between the domain modeling and software modeling realms with minimum loss of information and maximum simplicity. By mapping the OWL and UML constructs on a user interface, the knowledge in an ontology is automatically transformed into a UML class diagram. This class diagram can be understood by a developer, without any need for OWL knowledge.

In addition to the configurable construction of mappings, one other distinguishing feature in our approach is the handling of domain specific semantics captured in ontologies. Treating the transformation from OWL to UML as an ordinary XML transformation causes loss of information. Especially the semantics of restrictions and enumerations need special handling and interpretation. Our tool makes it available to use these constructs as the sources of the mappings. It also allows the user to activate some inferences on the ontologies. For example, the subclasses of an OWL class (as well as sub-properties of an OWL property) can be retrieved in a recursive way, whereas it is also possible to process only the direct subclasses of an OWL class.

We strived for simplicity with respect to tool's features. Consider, for example, designation of the visibility (public/protected/private) of attributes. It could be offered as a part of the user's options in specifying mappings. However, this could complicate the user interface and consistency checking slightly. We have followed a more straightforward approach, which is to set the visibility to "public" as a default for classes and attributes, and to let the user set the visibility on the UML class diagram whenever necessary.

In this research we utilized ontology based model driven software engineering practices for simulation development. This paper presents both an example of an ontology based simulation development and the application of state of the art software engineering practices in simulation domain.

We believe it is worthwhile to try the user-guided approach in other transformation problems as well. In cases where the source and target metamodels are well-structured around comparable basic concepts, and the uses of the target models are clearly defined there seems to be a potential to achieve the efficacy of transformations at the expense of negligible effort in constructing them.

Intelligent agents that use domain ontologies like TSONT to reason over composability of services for M&S can be envisioned (see [24]). Ontology driven service oriented trajectory simulation using TSONT is a future work.

7. REFERENCES

[1] OMG, Object Management Group, http://www.omg.org

[2] MDA, OMG Model Driven Architecture, http://www.omg.org/mda

[3] R.Prieto-Diaz, 1990, “Domain Analysis: An Introduction”, ACM SIGSOFT Software Engineering Notes, ACM Press.

[4] W. Hesse, 2005, “Ontologies in the Software Engineering Process”, EAI 2005 - Proceedings of the Workshop on Enterprise Application Integration, Berlin.

[5] R.A. Falbo, G. Guizzardi, and K.C. Duarte, 2002, “An Ontological Approach to Domain Engineering”, International Conference on Software Engineering and Knowledge Enginnering, Ischia, Italy.

[6] J.A. Miller and P.A. Fishwick, 2004, “Investigating Ontologies for Simulation Modeling”, Proceedings of the 37th Annual Simulation Symposium (ANSS’04), Arlington, VA, USA.

[7] OWL Web Ontology Language Overview, http://www.w3.org/TR/2004/REC-owl-features-20040210

[8] M. Horridge, H. Knublauch, A. Rector, R. Stevens, C. Wroe, 2004, “A Practical Guide To Building OWL Ontologies Using The Protégé-OWL Plugin and CO-ODE Tools”, The University of Manchester, Stanford University.

[9] U. Durak, H. Oguztuzun, and K. İder, 2006, “An Ontology for Trajectory Simulation”, Proceedings of the 2006 Winter Simulation Conference, Monterey, CA, USA.

[10] MOF, Meta-Object Facility, http://www.omg.org/mda/specs.htm#MOF

[11] UML, Unified Modeling Language Infrastructure, v2.1.1, http://www.omg.org/cgi-bin/doc?formal/07-02-06

[12] EMF, Eclipse Modeling Framework Project, http://www.eclipse.org/modeling/emf/?project=emf

[13] IODT, Integrated Ontology Development Toolkit, http://www.alphaworks.ibm.com/tech/semanticstk

[14] EODM, EMF Ontology Definition Metamodel, http://www.eclipse.org/modeling/mdt/?project=eodm#eodm

[15] XMI, XML Metadata Interchange, http://www.omg.org/technology/documents/formal/xmi.htm

[16] U. Durak, H. Oguztuzun, and K. İder, 2008, “An Ontology Based Trajectory Simulation Framework”, Journal of Computing and Information Science in Engineering, Vol 8.

[17] D.K. Pace, 2000, “Ideas About Simulation Conceptual Model Development”, Johns Hopkins APL Technical Digest, vol. 21, no. 3.

[18] R.S. Pressman, 2009, *Software Engineering: A Practitioner's Approach*, 7th Edition, McGraw-Hill.

[19] Rational Rose XDE Developer for Java, http://www-306.ibm.com/software/awdtools/developer/java/

[20] ArgoUML, http://argouml.tigris.org/

[21] D. Gasevic, D. Djuric, and V. Devedzic, 2006, “Model Driven Architecture and Ontology Development”, Springer, pp. 164-165.

[22] Y. Pan, G. Xie, L. Ma, Y. Yang, and Z. Qui, 2005, “An MDA-Based System for Ontology Engineering”, IBM research report: IBM TR RC23795.

[23] MTF, Model Transformation Framework, http://www.alphaworks.ibm.com/tech/mtf

[24] A. Tolk, 2006, “What comes after the Semantic Web, PADS Implications for the Dynamic Web”, Proceedings of the 20th ACM/IEEE/SCS Workshop on Principles of Advanced and Distributed Simulation (PADS 2006), Singapore, May 2006, pp. 55-62.

B.4 Tool Support for Transformation from an OWL Ontology to an HLA Object Model

Ozer Ozdikis, Umut Durak and Halit Oguztuzun. 2010. Tool support for transformation from an OWL ontology to an HLA Object Model. In *Proceedings of the 3rd International ICST Conference on Simulation Tools and Techniques*, ICST, Malaga, Spain.

Tool Support for Transformation from an OWL Ontology to an HLA Object Model

Özer ÖZDİKİŞ
OYAK Technology
Teknokent, Galyum Blok, Kat:1,
No:23 ODTU-Ankara TURKEY
+90 535 922 46 40
oozdikis@oytek.com.tr

Umut DURAK
TUBITAK-SAGE
PK.16 06261 Mamak
Ankara TURKEY
+90 312 590 91 76
udurak@sage.tubitak.gov.tr

Halit OĞUZTÜZÜN
Middle East Technical University
Computer Engineering Dept.
Ankara TURKEY
+90 312 210 55 87
oguztuzn@ceng.metu.edu.tr

ABSTRACT
Designing simulation architectures based on domain models is a promising approach. Tools to support transformation of formalized domain models to design models are essential. Ontology languages offer a way of formally specifying the domain knowledge. We adopt a user-guided approach to model transformation, where the source is an OWL ontology and the target is an HLA Object Model, in particular, a federation object model (FOM). This paper presents a flexible transformation tool that enables the user to define transformations in terms of mappings from OWL constructs to HLA Object Model Template (OMT) constructs. The overall objective is to facilitate ontology-based model-driven development in distributed simulation.

Categories and Subject Descriptors
I.6.7 [**Simulation Support Systems**]

General Terms
Design

Keywords
Ontology based simulation, model driven development, High Level Architecture, object models.

1. INTRODUCTION
In the context where distributed simulation architectural design and model driven development meet, the issue of transformation of domain models to platform-specific object models arises. A domain model, which captures knowledge from an area of interest, is an outcome of domain analysis. The approach that is based on the use of model transformations from a domain model to design models of varying levels of detail, and finally to code is known as Model Driven Development (MDD) or Model Driven Engineering (MDE). OMG's Model Driven Architecture (MDA) [10] and ISIS' Model Integrated Computing (MIC) [11] are particular manifestations of MDD/MDE.

Ontologies have recently gained popularity for representing domain knowledge for ease of both human understanding and machine processing [17][6]. Using ontologies as domain models is known as ontology based domain engineering [4]. In applying ontology based domain engineering to simulation development, we envision to derive reusable simulation components and artifacts.

A domain model ideally reflects all the stakeholders' views of the problem area. Further, we hold that tool-supported methodologies are required to bring the simulation developer's point of view into life. Our present focus is on tool support for flexible transformations from a domain ontology, which can be regarded as a representation of a simulation conceptual model [16], into an HLA object model.

OWL is an ontology language [15][7], which enjoys popularity due to "semantic web". When it comes to transforming an available OWL ontology into some target model, the "one size fits all" approach does not work. Because every domain model may require a different transformation procedure depending on the context, data types, conventions, and even the personal preferences of the simulation developer.

The tool provides a user interface to configure mappings from OWL constructs to HLA OMT constructs. Then, mapping definitions are applied on a given OWL ontology (formalizing a domain model), and consequently an HLA Object Model, in the form of an XML document [8], is produced.

1.1 Related Work
France and Rumpe [5] discuss how modeling techniques can be effectively leveraged during software development. Moreover, they note that, "there is a growing realization that MDE requires semantic-based manipulation of models". We believe our work takes some steps along this direction.

Tolk in [19] draws attention of the HLA based distributed simulation community to MDA and points out that employing MDA will enable HLA implementers to improve their products by making better use of the commercial technology.

SIMUTools 2010 March 15–19, Torremolinos, Malaga, Spain.

Digital Object Identifier: 10.4108/ICST.SIMUTOOLS2010.8678
http://dx.doi.org/10.4108/ICST.SIMUTOOLS2010.8678

Miller and Fishwick [12] identify the potential benefits of ontologies for modeling and simulation. In [20], Tolk also stresses the importance of conceptual data models, which can be parts of ontologies, in simulation development. He argues that in a simulation consisting of several participating systems, ontologies can be used to describe their services and information exchange capabilities to satisfy M&S composability and interoperability.

The work by Rathnam and Paredis [18] also addresses the use of ontologies in constructing HLA-based distributed simulations. In their work, the object models, namely, the FOM of a federation and the SOMs of the federates, are represented as ontologies. The mappings between individual SOMs and the FOM are also represented as an ontology. By means of these mappings, the reusability of existing federates in a new federation is facilitated. That approach requires the user design his ontology specific to HLA standards. In our work the ontology captures the simulation conceptual model in a more abstract way, in that it is not specific to HLA or any other simulation standard. HLA-specific information is provided by the transformations from the ontology to the object model.

In a previous study [14], we achieved to provide tool support for user-guided model transformation from ontology to the object oriented design for the simulation software, in the form of a UML class diagram. This present effort is built upon the premise that the domain knowledge that is standardized in the form of a common ontology can be utilized to derive a representation of the information shared among the participants in a distributed simulation.

1.2 Background

This effort builds up a weak analog to the levels of abstraction that are identified in OMG's MDA [10] while developing a tool support for model driven simulation development. MDA presents the abstraction levels of system development as follows: The computation independent viewpoint as the first abstraction level, focuses on the environment in which the system of interest will operate in and on the required features of the systems [5]. The platform independent viewpoint focuses on the aspects of system features that are not likely to change from one platform to another. We expect ontologies to possess both computation independent or platform independent viewpoints depending on their design purpose and content. The next step is platform specific viewpoint which is regarded as the last level of abstraction before the executable code. It specifies how that system utilizes a particular platform.

The idea is that domain knowledge which is captured at a conceptual level will be used to generate the models towards the executable assets as automated as possible utilizing model transformation practices. This effort tries to build a tool to allow the simulation engineer to guide the transformation from conceptual model which is represented by an OWL ontology, to an asset towards executable code, which is an Object Model. Object Models are regarded as HLA specific interface models which then can also be transformed to Federation Design Data [21].

Our transformation process can be located in reference to the four-layer metamodeling hierarchy of OMG's Meta Object Facility (MOF) [13]. MOF is defined as the extensible model driven integration framework for defining, manipulating, and integrating metadata and data in a platform independent manner. It provides a meta-metamodel at the top level, which is called M3 layer of the four-layer metamodeling hierarchy. Any M3-layer meta-metamodel can be used to define more specific metamodels at M2 layer, such as the HLA Object MetaModel (HOMM). A fully-fledged metamodel for the HLA Object Model is provided as a part of the Federation Architecture Metamodel (FAMM) [21]. We have used a scaled down version of HOMM. FAMM employs metaGME, the meta-metamodel provided in GME, a (meta)modeling environment supporting MIC. Object Models conforming to the HOMM (thus, to the HLA OMT standard) are at layer M1. Finally, the M0 layer includes the objects and interactions created during federation execution as instantiations of the Object Model.

A specific ontology can be viewed as conforming to a metamodel (which plays the role of a grammar for an ontology language, such as OWL). Our ontology modeling hierarchy is based on Eclipse Metamodel Framework (EMF) [2]. The meta-metamodel at M3 layer in EMF is called Ecore. IBM implemented an Integrated Ontology Development Toolkit (IODT) for ontology driven development built on EMF [9]. IODT includes a library called EMF Ontology Definition Metamodel (EODM) which is an implementation of the OMG's Ontology Definition Metamodel [3]. EODM has OWL parsing, serialization, and reasoning features.

In the transformation, we have an ontology model as our source and the HLA Object Model as our target. Our approach facilitates the definition of the mappings between the EODM and HOMM at M2 layer. These mappings are applied to a given OWL ontology to generate an HLA Object Model. The modeling layers which are used in this transformation are shown in Figure 1.

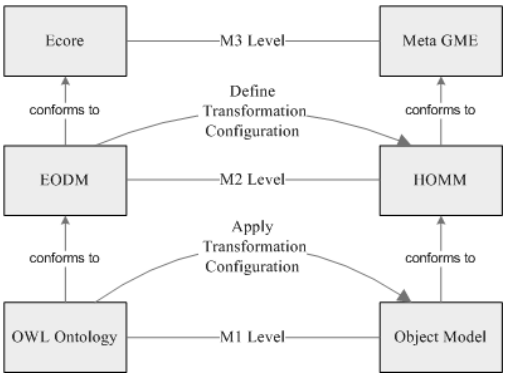


Figure 1. Relations between the modeling layers

In the following sections, available OWL constructs of the source and OMT constructs of the target are introduced. The tool that was developed to configure the mapping from source to target is presented. Finally, our ongoing work on a case study and future work are commented upon.

Digital Object Identifier: 10.4108/ICST.SIMUTOOLS2010.8678
http://dx.doi.org/10.4108/ICST.SIMUTOOLS2010.8678

2. THE OWL-TO-OBJECT MODEL TRANSFORMATION

Our tool lets the user configure the transformations as appropriate. A transformation configuration is composed of mapping groups, mappings and constraints.

A mapping group is a collection of mappings from some specific OWL constructs to some OMT constructs. In other words, a mapping group includes the specification of source OWL constructs and mappings to apply on these constructs. Source constructs can be OWL classes or OWL properties. The user can define constraints on the source OWL constructs, so that the mappings in the mapping group are applied only on the desired subset of source constructs.

Following the description of the source constructs, the user must specify, in terms of mappings, how to use them in the transformation. Depending on the source-target combinations, the user can define four types of mappings in a mapping group: to Object Class, to Attribute, to Interaction Class and to Parameter. A mapping is the prescription of how the target OMT construct should be built using the source OWL construct.

Figure 2 shows a screenshot from the tool. It enables the user to define several mapping groups with several mappings inside. The boxes labeled OC, IC, AT and PR represent four different mapping types regarding OMT constructs.

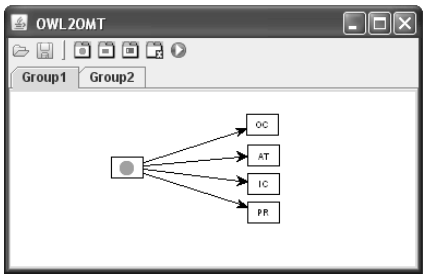


Figure 2. Overview of the UI

Constraints can be defined to restrict the entities to be evaluated in a specific mapping group or mapping. Constraints are actually condition-value pairs applied on an OWL construct. While evaluating an OWL construct in a mapping group or mapping, these condition-value pairs are used to check if that construct is selected and should be processed. As an example, if the user wants to define some mappings on a specific subset of source OWL objects, he must define a new mapping group, then define a constraint for that mapping group to specify the interested OWL objects, and finally define his mappings in that mapping group so that they are applied only on the specified source OWL objects.

The last step of the transformation is the validation of the resulting model. Since this transformation is a user-guided transformation, there may be inconsistencies, for example, a reference to a Dimension that actually does not exist in the object model. Details of validation are explained in the forthcoming sections.

2.1 Available OWL Constructs at the Source

An OWL ontology involves Classes, Object Properties, and Datatype Properties. A Class has a name and possibly super classes. A Class may be defined as an intersection of, union of or complement of other classes. Further, a class may have different types of restrictions, namely, MinCardinalityRestriction, MaxCardinalityRestriction, CardinalityRestriction, AllValuesFromRestriction, SomeValuesFromRestriction and HasValueRestriction. These restrictions define the values (Restriction.Value) that the Class must take for a Property (Restriction.Property). A Class may also be an enumeration class, which includes the list of individuals that are the members of the class. A Property has a name, domain and range information, and possibly super properties. Our tool lets the user use definitions of Classes, Object Properties, and Datatype Properties as a source for the mappings to OMT.

2.2 Available OMT Constructs at the Target

According to the IEEE 1516 Standard [8], an OMT model consists of the definitions of Object Classes and their Attributes, Interaction Classes and their Parameters, Dimensions, Datatypes, Transportations, Switches, Time, Synchronizations and User Supplied Tags. Our primary concern here is the creation of Object Classes, Attributes, Interaction Classes and Parameters. These constructs may have references to Datatypes, Dimensions and Transformations and if these referenced constructs do not exist in the target Object Model, new Datatype, Dimension and Transportation definitions will be introduced with default properties through the transformation process. Details of these OMT constructs are expected to be edited with an OMT Editor by the user after the transformation. Switch, Time, User Supplied Tags and Synchronization definitions are ignored in this process.

The class hierarchy for Object Classes and Interaction Classes are represented by *subClassOf* properties in our OMT model. While serializing the model into FOM file, these *subClassOf* properties are replaced with the nested class definitions.

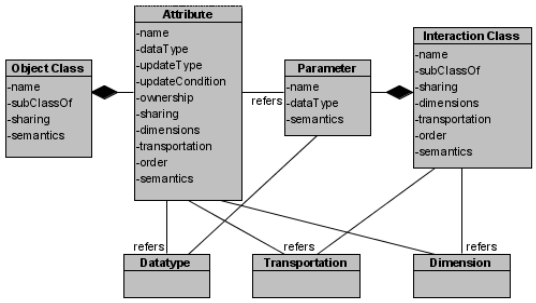


Figure 3. Target OMT Constructs

Figure 3 shows the relationships between the OMT objects and their properties handled during a transformation. Some properties are allowed to take a value from a predefined value set (*sharing* property of an Object Class can be set to "Publish", "Subscribe", "PublishSubscribe" or "Neither"), some can take any String (like the *name* of an Object Class) and others must refer to an existing OMT object in the model (*dataType* property of an Attribute must be the name of an existing Datatype object).

Digital Object Identifier: 10.4108/ICST.SIMUTOOLS2010.8678
http://dx.doi.org/10.4108/ICST.SIMUTOOLS2010.8678

2.3 Mappings from OWL to OMT

Our tool provides an interface to the user to define mappings between the available OWL constructs at the source and the OMT constructs at the target, which were defined above. Mappings can be classified into 4 types regarding the target constructs. These are

- Mappings to Object Classes
- Mappings to Attributes
- Mappings to Interaction Classes
- Mappings to Parameters

Our tool lets a mapping read the values of the properties of above OMT constructs from the ontology. The user can also fix the values in the transformation configuration. For instance, he can either say "The value for *subClassOf* property of an Object Class will be taken from the name of the super class of corresponding OWL Class" or "Attributes of Object Classes whose name begin with 'X' will have *datatype* 'HLAboolean'".

The definitions of these 4 OMT objects may have references to Datatype, Dimension and Transportation definitions that do not exist in target Object Model by default. These cases are handled by adding the definitions for these new OMT constructs with their default properties. For example, if the transformation results in an Interaction Class with its *transportation* 'X', a Transportation object with *name* 'X' is added to the resulting Object Model. Details of this Transportation instance are supposed to be configured later manually by the user.

During the transformation process, mappings to Object Classes and Interaction Classes are resolved first. As will be explained in the following sections, new Attributes and Parameters are introduced to the target model during the resolution of mappings to Object Classes and Interaction Classes, respectively. Mappings to Attributes and Parameters are used to set their further properties like *datatype* or *sharing*.

2.3.1 Mappings to OMT Object Classes

This mapping type enables the user to define new Object Classes in the target model. The *name* of the Object Class is taken from the name of the OWL Class/Property in the source. While creating new Object Classes, user can also configure mappings for the properties of these Object Classes. In other words, he can configure how to set the *subClassOf*, *sharing* and *semantics* of the related Object Classes. Each Object Class can have at most one super class, and the name of this superclass is represented in a *subClassOf* property in the target model. The name of the super class can be taken from the source ontology constructs depending on the mapping configured by the user as shown in Figure 4.

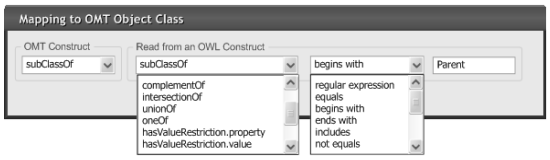


Figure 4. Object Class Mapping for subclass relationship

As an example, the configuration in Figure 4 will be evaluated as follows: OWL Classes/Properties in the input ontology are traversed one by one according to the constraint definitions for the mapping group which includes this mapping. For each valid OWL Class/Property in the source, an Object Class with same *name* is created in the target Object Model. Moreover, if an OWL Class/Property named "C" is defined to be the subclass/subproperty of OWL Class/Property named "ParentC", then Object Class "C" will be the subclass of Object Class "ParentC" in the resulting Object Model.

The other property of an Object Class is *sharing*. *Sharing* can either be taken from an OWL construct in the source or set to one of the possible values in a choice list. Figure 5 shows the configuration panel to define how to set the *sharing* property of related Object Classes.

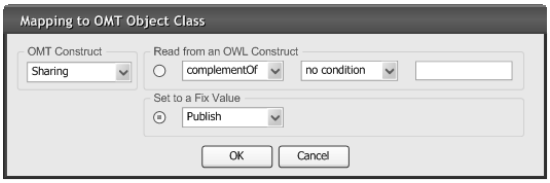


Figure 5. Object Class Mapping for Sharing

Similar to the *sharing* property, *semantics* can either be taken from an OWL construct in the source or set to some fix value. Figure 6 shows an example, which also illustrates the constraint definitions. The mapping configuration in Figure 6 is processed as follows: if the OWL Class in the source has an OWL HasValueRestriction definition on an OWL Property named "description", the value of this specific Restriction will be set as the *semantics* of the corresponding Object Class.

Figure 6. Object Class Mapping for Semantics

Object Classes can have Attributes. In the mapping configuration for Object Class, user can define where to get the attribute names for the corresponding Object Classes. If the user configures the mapping for the attribute names, new Attribute objects with desired names are generated in the target model. In this same mapping, user can also set the properties of Attribute objects, i.e. *dataType*, *updateType*, *updateCondition*, *ownership*, *sharing*, *dimensions*, *transportation*, *order* and *semantics* as shown in Figure 7. However our tool does not allow getting the values of Attribute properties from the source OWL constructs in this mapping panel, instead their values are set to some desired fix values. If the user wants to get Attribute property values from the OWL ontology, he has to define a new "Mapping to Attributes" as explained in the following section. The required attribute mapping flexibility is provided in that mapping type.

Digital Object Identifier: 10.4108/ICST.SIMUTOOLS2010.8678
http://dx.doi.org/10.4108/ICST.SIMUTOOLS2010.8678

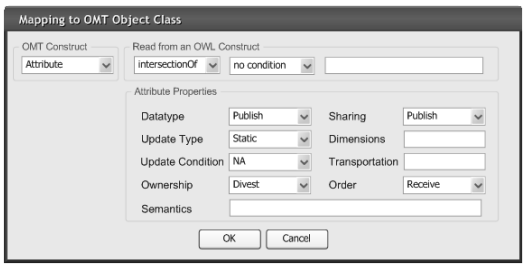


Figure 7. Object Class Mapping for Attributes

2.3.2 *Mappings to OMT Attributes*

There may be cases where some Class or Property in the ontology defines an Attribute with its properties. This mapping type is used to set the properties of Attributes which were created during the configuration of "Mappings to Object Classes". The transformation for this type of mapping works as follows: OWL Classes/Properties in the input ontology are traversed one by one according to the constraint definitions for the mapping group. For each selected OWL Class/Property named "C", an Attribute with name "C" is searched in the target model. For each Attribute named "C", the OWL construct defined in the mapping is used to feed the property values for this Attribute. Figure 8 shows a case where the *datatype* of defined Attributes are taken from the property name of the MaxCardinalityRestrictions defined for the corresponding OWL Class.

Figure 8. Attribute Mapping for possible Attribute properties

The precondition for this mapping is that the correct Attributes have already been defined for desired Object Classes. This is achieved by the Attribute configurations in Object Class mappings. Thus, "Mapping to Attributes" is just a matter of feeding the property values to the previously defined Attributes of Object Classes. One additional feature in this mapping is that the name of the owner Object Class can be bound to a constraint in the mapping.

2.3.3 *Mappings to OMT Interaction Classes*

The mappings for Interaction Classes and Parameters are similar to the mappings for Object Classes and Attributes. With this mapping type, new Interaction Classes are added to the target model. Properties of the Interaction Classes, namely *subClassOf, sharing, dimensions, transportation, order, and semantics* are also configured by either using the source OWL constructs or selecting values from choice lists. Moreover, if the Interaction Class needs to have Parameters, user can configure this mapping to create Parameters with desired names for the corresponding Interaction Classes. In this mapping window, user can also choose a *dataType* for the Parameters from a choice list. If the *datatype* for the Parameters are to be taken from some OWL constructs, the user has to configure a "Mapping to Parameters".

2.3.4 *Mappings to OMT Parameters*

Just like for the "Mappings for Attributes", this mapping requires that the Parameters are already defined in the target model through the performance of "Mappings for Interaction Classes". This mapping type enables the user to set the OWL constructs to feed the *datatype* and *semantics* for desired Parameters. During the execution of this mapping, each OWL class/property in the source ontology is traversed and if a Parameter with the same OWL class/property name is found, its *datatype and semantics* are set with the OWL construct according to the mapping configuration. Moreover user can define a constraint on the name of the encapsulating Interaction Class to apply this mapping.

2.4 Validation of the Model

The last step of the transformation is Object Model validation. The reason for this step is that the resulting model may not always be consistent. Especially if the model constructs are to be taken dynamically from the source ontology, the values set to these constructs may not be in the allowed range or referred objects may not exist in the target object model. The validation checks our tool currently applies include the following:

- **Enumerations:** Some OMT constructs (namely *sharing, updateType, ownership* and *order*) may get only some restricted specific values. For example, if the transformation sets the *sharing* property of an Object Class to a value other than "Publish", "Subscribe", "PublishSubscribe" or "Neither", this would not be a valid FOM.
- **Class hierarchy:** An Object Class or an Interaction Class cannot have multiple super classes.
- **Uniqueness:** There cannot be two OMT constructs of the same type with the same name.
- **New OMT constructs:** Transformation may result in references to Datatype, Transportation or Dimension objects which do not exist in FOM. These constructs are introduced with default property values.
- **Dependent properties:** The value of a property may depend on the value of another property of an OMT construct. For example, if *dataType* of an Attribute is "NA", its *updateType*, *updateCondition* and *dimensions* must also be "NA" and a *transportation* and *order* must be specified for that Attribute.

3. DISCUSSION

The specific contribution of our work is a tool for ontology based simulation design. We introduce a user-guided transformation process to bridge the gap between the domain modeling and simulation software modeling realms with minimum loss of information and maximum simplicity. By mapping the OWL and HLA Object Model constructs on a user interface in a point-and-click fashion, the knowledge captured in an ontology is automatically transformed into a FOM. Once the transformations are defined, subsequent updates to ontology, in so far as they do

Digital Object Identifier: 10.4108/ICST.SIMUTOOLS2010.8678
http://dx.doi.org/10.4108/ICST.SIMUTOOLS2010.8678

not disturb the existing input-output patterns, are reflected to the target FOM without further user involvement. This FOM can be then used in an HLA simulation environment without any need for OWL knowledge.

An ongoing case study attempts to generate a FOM from the Trajectory Simulation Ontology (TSONT) [1]. The FOM will be for a federation involving simulation of some munition trajectories. TSONT essentially captures the trajectory simulation domain knowledge, including mathematical models, and specifies the functionality required to carry out a simulation. Currently, the generation of FOM is manual. Our goal is to let the user configure a transformation to automatically generate the same FOM.

Current tool design aims at generation of four main OMT constructs, namely Object Classes, Attributes, Interaction Classes and Parameters. As explained in Section 2.2, new Datatypes, Transportations and Dimensions are created with default properties if necessary. In future releases of our tool, we are planning to introduce new mapping types to let the user set the properties of Datatypes, Transformations and Dimensions using the information in the source ontology.

It is desirable for the tool to be able to read the source data from multiple ontology files. Multiple ontologies may account for multiple domains involved in a complicated federation scenario. These data sources may need to be combined and interpreted to generate a federation object model.

There are some structural differences between an ontology and an HLA Object Model. For example, an OWL class may have multiple superclasses, while an Object Class in FOM may have only one superclass. If the user configures the mappings which somehow result in an Object Class with multiple superclasses, only the last superclass assignment becomes effective.

4. REFERENCES

[1] Durak,U., Oguztuzun,H., and İder,K. 2006 An Ontology for Trajectory Simulation. Proceedings of the 38th Winter Simulation Conference, Monterey, CA, USA

[2] EMF, Eclipse Modeling Framework Project, http://www.eclipse.org/modeling/emf/?project=emf

[3] EODM, EMF Ontology Definition Metamodel, http://www.eclipse.org/modeling/mdt/?project=eodm#eodm

[4] Falbo, R.A., Guizzardi, G., and Duarte, K.C. 2002 An Ontological Approach to Domain Engineering. International Conference on Software Engineering and Knowledge Enginnering, Ischia, Italy.

[5] France, R., and Rumpe, B. 2007 Model-driven Development of Complex Software: A Research Roadmap. Proceedings of the Conference on Future of Software Engineering, (May 23-25, 2007), p.37-54.

[6] Hesse, W. 2005 Ontologies in the Software Engineering Process. EAI 2005 - Proceedings of the Workshop on Enterprise Application Integration, Berlin.

[7] Horridge, M., Knublauch, H. , Rector, A., Stevens, R., and Wroe, C. 2004 A Practical Guide To Building OWL Ontologies Using The Protégé-OWL Plugin and CO-ODE Tools, The University of Manchester, Stanford University

[8] IEEE Std 1516.2-2000, 2001, IEEE standard for modeling and simulation (M&S) high level architecture (HLA) - object model template (OMT) specification. http://ieeexplore.ieee.org/xpl/standardstoc.jsp?isnumber=19791

[9] IODT, Integrated Ontology Development Toolkit, http://www.alphaworks.ibm.com/tech/semanticstk

[10] Kleppe, A., Bast, W. and Warmer, J. B. 2003. MDA Explained, the Model Driven Architecture: The Model Driven Architecture: Practice and Promise. 2nd Ed. Addison-Wesley, Boston.

[11] MIC, Model Integrated Computing, http://www.isis.vanderbilt.edu/research/MIC

[12] Miller, J.A., and Fishwick, P.A. 2004 Investigating Ontologies for Simulation Modeling. Proceedings of the 37th Annual Simulation Symposium (ANSS'04), Arlington, VA, USA.

[13] MOF, Meta-Object Facility, http://www.omg.org/mda/specs.htm#MOF

[14] Ozdikis, O., Durak, U. and Oguztuzun,H. 2009 User Guided Transformation for Ontology Based Simulation Design. 2009 Summer Computer Simulation Conference, Istanbul, Turkey.

[15] OWL Web Ontology Language Overview, http://www.w3c.org/TR/2004/REC-owl-features-20040210

[16] Pace, D.K . 2000 Ideas About Simulation Conceptual Model Development. John Hopkins APL Technical Digest, 21, 3.

[17] Prieto-Diaz, R. 1990 Domain Analysis: An Introduction. ACM SIGSOFT Software Engineering Notes, ACM Press.

[18] Rathnam, T., and Paredis, C.J.J. 2004 Developing federation object models using Ontologies. Proceedings of the 2004 Winter Simulation Conference, Washington, DC, USA

[19] Tolk, A. 2002 Avoiding another Green Elephent – A Proposal for the Next Generation HLA based on the Model Driven Architecture. 2002 Fall Simulation Interoperability Workshop, Orlando, FL, USA.

[20] Tolk,A., and Turnitsa,C.D. 2007 Conceptual modeling of information exchange requirements based on ontological means. Proceedings of the 39th Winter Simulation Conference, Washington D.C.

[21] Topçu, O., Adak, M. and Oğuztüzün, H. 2008 A metamodel for federation architectures. ACM Transactions on Modeling and Computer Simulation, 18,3, (July 2008), 10:1-10:29. DOI=http://doi.acm.org/10.1145/1371574.1371576

Digital Object Identifier: 10.4108/ICST.SIMUTOOLS2010.8678
http://dx.doi.org/10.4108/ICST.SIMUTOOLS2010.8678

B.5 Model Integration Workflow for Keeping Models up to Date in a Research Simulator

Torsten Gerlach, Umut Durak and Jurgen Gotschlich. 2014. Model integration workflow for keeping models up to date in a research simulator. In *Proceedings of the 4th International Conference on Simulation and Modeling Methodologies, Technologies and Applications (SIMULTECH)*, SCITEPRESS, Vienna, Austria, 125-132.

Model Integration Workflow for Keeping Models up to Date in a Research Simulator

Torsten Gerlach, Umut Durak and Jürgen Gotschlich
Institute of Flight Systems, German Aerospace Center (DLR), Lilienthalplatz 7, Braunschweig, Germany
{torsten.gerlach, umut.durak, juergen.gotschlich}@dlr.de

Keywords: Model Integration, Flight Simulators, Model based Design and Development, Simulink Coder.

Abstract: Flight simulators can be categorised as research simulators, engineering simulators and training simulators. Research simulators can be introduced as both test beds for flight simulator research and computational tools for flight systems and human factors research. While engineering simulators are utilised for systems development, training simulators are used for flight training. The models that are used in training simulators and also in engineering simulators are more mature and stable. On the other hand, the models in research simulators are subject to a constant change. While Model Based Design and Software Development has brought us agile model development workflows, so that modellers can update their models more easily, it came up with some serious systems integration and testing problems, so systems developers need to establish mechanisms to tackle frequent behaviour and interface changes. DLR's Institute of Flight Systems (FT) has a long tradition in flight research and simulation of various flight vehicles. Currently a modern research simulator facility is being operated at DLR Braunschweig –AVES (Air Vehicle Simulator). AVES is designed such that interchangeable cockpits of rotorcraft (EC135) and airplanes (A320) can be operated on motion and fixed-base platforms according to the particular needs. 2Simulate is the enabling real-time simulation infrastructure of the AVES. This paper presents 2Simulate model integration workflow based on Mathwork's Simulink Coder.

1 INTRODUCTION

Till late 1920s, when Edward Link built one of the early examples of flight simulators, they have been important elements of aviation. These first examples which were known as *Blue Box,* were designed to train pilots for instrumented flight (Allerton, 2009). Before digital era, flight simulators became well accepted as training aids by many aircraft operators. Then as the fidelity of flight simulators increased, engineering standards to build flight simulators for flight training were developed.

As flight simulators became de facto tools in flight training, they were also started to be used in aircraft development. After 1980s, testing and validation of aircraft systems started to be done in engineering flight simulators. Thus, potentially dangerous and expensive flight tests could be avoided (Allerton, 1999).

In 80s, aeronautics research community was also using flight simulators for developing and experimenting advanced concepts. ATTAS Simulator from German Aerospace Center (DLR) (Saager, 1990) (Klaes, 2000), NASA Crew Vehicle Systems Research Facility in Ames Research Center (Sullivan & Soukup, 1996) and Visual Motion Simulation and Cockpit Motion Facility from Langley Research Center (Smith, 2000) were some of the first examples of research flight simulators. Some of the recent ones are Air Vehicle Simulator (AVES) of German Aerospace Center (DLR) (Duda et al., 2013), HELIFLIGHT from the University of Liverpool (White & Padfield, 2006), NASA Ames Vertical Motion Simulator (Advani et al., 2002) and SIMONA of the Delft University of Technology (Stroosma et al., 2003).

The organisation of a flight simulator is structured around the flight dynamics model. There may be various components that supports or works with flight dynamics model, like aerodynamics model, landing gears model, weather model, engine model and subsystem models. The architecture of these models varies from simulator to simulator. They can either be implemented as a single model or various models interacting in a tightly coupled manner. The other important components like

SIMULTECH 2014 - 4th International Conference on Simulation and Modeling Methodologies, Technologies and Applications

control loading, instructor station, motion system, visual system, instrument displays either provide inputs to these models or present their results to the pilot as cues (Allerton, 2009).

Research simulators have been used as the test beds for flight simulator, flight systems and human factors research. So, while the models that are used in training simulators and even in engineering simulators are more mature and stable, the models in research simulators are subject to a constant change.

Recent advances in Model Based Design and Software Development (MBDSD) have brought aeronautics community agile model development workflows. So that model development is integrated to product development employing mature code generation practices (Ruff et al., 2012). Models developed to design the products now became the bases for code generation to be deployed in the product.

For research flight simulators with MBDSD, the models that are built to study overall systems (e.g. Flight Dynamics Model) and subsystem (e.g. Flight Warning Computer Model) behaviour became the bases for generating code to be deployed in real time flight simulators. These models serve for researchers that exercise various aspects of aircraft in their desktop environments, and for simulator developers to simulate aircraft system and sub systems.

Research simulator developers need to establish mechanisms to tackle frequent behaviour and interface changes in models. And constant model changes in a research habitat can only be enabled with a model integration workflow in the systems development. But flight simulator literature lacks in reporting any efforts.

There are some recommended practices from the aerospace industry for model based flight systems design and development. Estrada et al. introduce best practices for developing DO-178 compliant software using Model-Based Design and Development (Estrada, R.G. et al., 2013). Miller presents automatic flight code generation practices in Northrop Grumman (Miller, 2007) and introduces a use case from desktop simulation to Hardware in the Loop testing. BAE Systems has a model based flight control systems development process (Fielding, 2010). Fielding presents a process starting from aerodynamic dataset generation to flight clearance of the aircraft. In this process he mentions the use of engineering simulators for model based flight control system design. Nixon states that in F-35 project MBDSD forced them to reinterpret traditional software development process for flight control systems (Nixon, 2004). He introduces Lockheed Martin Aeronautics practices of MBDSD.

On the other hand, there exists a vast amount of effort to develop integration workflows for their model based developed software component. In one of them (Guido and Thompson, 2008) from Mathworks, authors propose a workflow to develop software components to be integrated to Automotive Open System Architecture (AUTOSAR). AUTOSAR specifies the architecture to integrate functional applications over a hardware abstracting runtime environment in automotive electronic control units. The presented workflow enables modellers to develop an infrastructure compliant model development and seamless integration over a standard architecture.

Figure 1: DLR AVES.

DLR's Institute of Flight Systems (FT) has a long tradition in flight research and simulation of various flight vehicles. Currently AVES, a modern research simulator facility is being operated at DLR Braunschweig. AVES is designed such that interchangeable cockpits of rotorcraft (EC135) and airplanes (A320) can be operated on motion and fixed-base platforms according to the particular needs. 2Simulate is the enabling real-time simulation infrastructure of AVES. All simulator software components are integrated over this infrastructure. This effort adopts best practices from both aerospace and automotive industries. It tackles the model integration problem of research flight simulators by developing a model integration workflow for the indigenous simulator infrastructure, namely 2Simulate. The motivation is to contribute to flight simulator development by introducing a model integration workflow for institutionalizing MBDSD.

The paper presents the Mathwork's Simulink Coder based model integration workflow of 2Simulate infrastructure in AVES facility. This workflow provides the users of AVES a shortened time to simulator after they updated their models.

First the reader will be introduced to 2Simulate. Following the presentation of proposed model integration workflow, how the 2Simulate Model Control is designed to enable this workflow will be discussed. Sample model integration will then be provided to exemplify the concepts and technologies introduced.

2 2SIMULATE

AVES was developed based upon the strategy to employ reusable, flexible, standardized and properly validated software modules. 2Simulate is an overall simulation framework to facilitate integrating a wide range of models and simulation components like external devices, data recorders or image generators. (Gotschlich et al., 2014).

It is a C++ real-time distributed simulation framework which is composed of three components, namely 2Simulate Real-Time Framework (2SimRT), 2Simulate Model Control (2SimMC) and 2Simulate Control Center (2SimCC) (Figure 2).

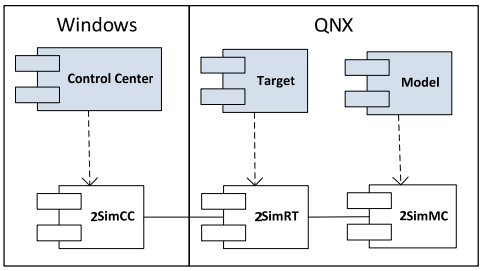


Figure 2: 2Simulate Components.

2SimRT is the core simulation framework of 2Simulate that provides deterministic scheduling and controlling of real-time tasks. It comes as libraries and API header files for Windows to support soft real time implementations like desktop simulators, or QNX to support hard real time implementations like full flight simulators. Hard real time use case is targeted in the scope of this paper. Any simulation application that is based on 2SimRT is called a Target. Targets possess various real-time tasks that are implemented utilizing the 2SimRT API. TSimModel is one of these real-time tasks used for integrating Simulink models. Tasks can be programmed using their pre- and post-initialization and pre- and post-process callbacks. 2SimRT also provides a Common Database to manage the data flow through the internal and external interfaces (Figure 3).

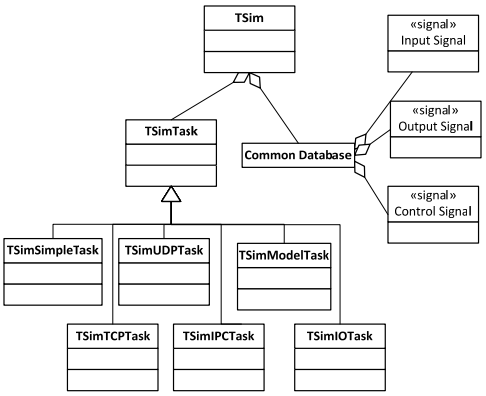


Figure 3: Main 2Simulate Classe.s

2SimMC is the enabler of model integration workflow. It is composed of 2Simulate Model Control Source (2SimMC-Source) that abstracts model interfaces for 2SimRT, and 2Simulate Model Control Scripts (2SimMC-Scripts) that includes Simulink Coder Target Language Compiler files (TLC files) to specify the 2Simulate target and m-files to conduct the code generation and build process.

2SimCC is the graphical user interface that is configured to a Control Center for specific needs. It is a Windows executable which can be customized via configuration files called 2SimCC project files. Control Center can run, pause or stop various Targets. Besides, it accesses the Target Data Dictionaries which can be defined as the data access mechanisms and enables presenting or editing Target data at runtime. It can also enable user management to define and enforce user access rights.

3 INTEGRATION WORKFLOW

Model integration workflow is triggered if any of the simulator models in the simulator are updated. As soon as the update is tested and verified in the modeling environment, which is Matlab/Simulink, the modeler would like to deploy it to its target. The process assumes that the modeler assures that the model is valid and correct.

Proposed workflow starts with a static model checking step before continuing to generating source code. Checking a model for modeling guidelines will provide a set of valuable information about what best practices or guidelines are violated. Thus it contributes to the quality of the model (Fey and

Stürmer, 2007). Measuring and assessing the quality of the model for code generation has various aspects including structured and automated testing, coverage analysis, complexity analysis, modeling guidelines (Stürmer and Pohlheim, 2012). Matlab Model Advisor is employed as a starting point in this step to check the mode for conditions and configurations that may lead to generation of inaccurate and inefficient code (The MathWorks, Inc., 2007) .

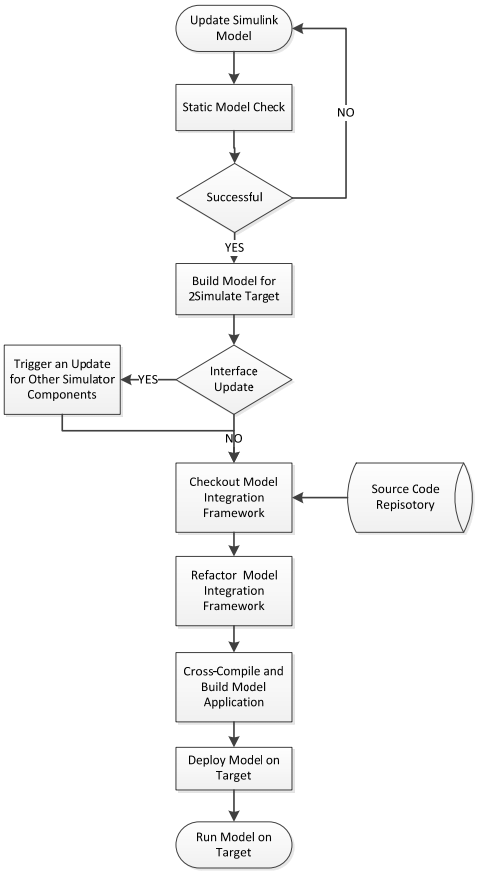


Figure 4: Model Integration Workflow.

As depicted in Figure 4, the next step is building the model for 2Simulate target. The Simulink model can be used with 2Simulate after it has been converted into C++ code using Mathworks Simulink Coder (The Mathworks, Inc., 2014a). A part of the Simulink Coder is the Target Language Compiler. It specifies the code generation (The Mathworks, Inc., 2014b) utilizing so called system target files, which can be customized for specific needs. 2Simulate has such a set of customized system target files. They embed 2SimMC into the model code during the code generation. Thus, an auto-generated model is readily available for 2Simulate model task. At this step, these set of target files are employed. The details of 2Simulate system target files will be presented in the next section.

The changes in the model interface are traceable over the signal specifications for the model generated, while code generation process. It is almost clear that any change in the model interface will cause an update in the other simulator components that depend on these signals. So the next step of the integration workflow is to trigger an update process for the other components if any change in the model interface is identified.

The next two steps in the process aims at preparing the model application source project. At first Model Integration Framework is checked out from the source repository. This framework is a wrapper for the generated model code. It creates a 2SimRT target using the generated model code. This wrapper code is refactored automatically for model specific parameters. As an example, the solver step size of the model is set as the frequency of the model task in the Model Integration Framework code. After refactoring model application code is ready for compilation.

The rest of the process is to cross-compile the application code for the QNX target and deploy the generated image to the target system. At the end of the workflow, the updated model becomes readily runnable at the target system.

4 2SIMULATE MODEL CONTROL

In this section, components of 2SimMC, 2SimMC-Scripts and 2SimMC-Source will be introduced.

There are two types of scripts in 2SimMC-Scripts. A Matlab script *TSimModelBuilder.m* is in charge for Matlab automation for every step that is depicted in Figure 4. And TLC files are used to specify the 2Simulate target.

TSimModelBuilder.m makes use of Matlab command line utilities for controlling Model Advisor, Simulink Coder and calling some external executables for source control, cross-compilation and secure shell. It also conducts refactoring in the Model Integration Framework code employing file and string manipulation utilities of Matlab. Below is a representative code extract from *TSimModelBuilder.m*

```
function status = TSimModelBuilder(ModelName)

%% Check Model for Errors

% Get model advisor handle
mAd = Simulink.ModelAdvisor.getModelAdvisor(ModelName);
...

%% Build Simulink Model
...
% Run rtwbuild
status.rtwbuild = rtwbuild(ModelName);

%% Get Model Integration Framework
cmdStr = ['svn export  svn://.../2SimulateMI ./'...
    ,ModelName, '_2Simulate/2Simulate'];
status.MIFExport=dos(cmdStr);
...
%% Refactor Model Integration Framework

sourceFile = [ModelName,'_2Simulate\mdlTarget.cpp'];
fid  = fopen(sourceFile,'r');
f=fread(fid,'*char')';
fclose(fid);
...
%set step size
stepSize = num2str(get_param(ModelName, 'FixedStep'));
f = strrep(f,'<model_steptime>',stepSize);
...
%% Build QNX Project
...
cmdStr=['eclipsec ... -project mdlTarget -target rebuild'];
status.QDEBuild = dos(cmdStr);
%% Deploy the image
%winscp
cmdStr=['winscp ./mdlTarget user@192.168.0.1:/'];
status.Deployment = dos(cmdStr);
```

Simulink Coder allows its users to select a target for code generation. Target Language Compiler on the other hand, provides capabilities to specify targets through customizing the generated code to produce platform or application specific code. It transforms *model.rtw,* the intermediate form of Simulink block diagram into C or C++ code. Code generation is controlled by TLC files. TLC files have uses a syntax like Perl or other scripting languages, augmented with data handling capabilities of Matlab (The Mathworks, Inc., 2014b). One can create and modify the generated code, generation time data processing with TLC directives and accessing model structure captured in *model.rtw*. It provides looping, file I/O, scoping type powerful scripting tools.

For 2Simulate a target specification called *grt_2Simulate* is implemented by 2SimMC-Scripts TLC files. These files extend generic real-time target provided by Simulink Coder. The top level entry point is *grt_2Simulate.grt*. As presented in Figure 5, it first calls *codegenentry.tlc* to generate model code and then calls all eight 2Simulate TLC files to generate 2SimMC-Component code.

2SimMC-Component code is composed of sources for a 2SimRT task, model, data dictionary, model defines and specifications for input and output signals. Task and model TLC files extend

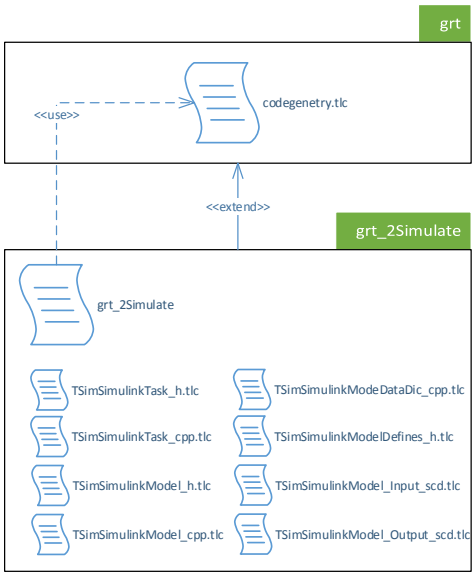


Figure 5: 2Simulate System Target File Structure.

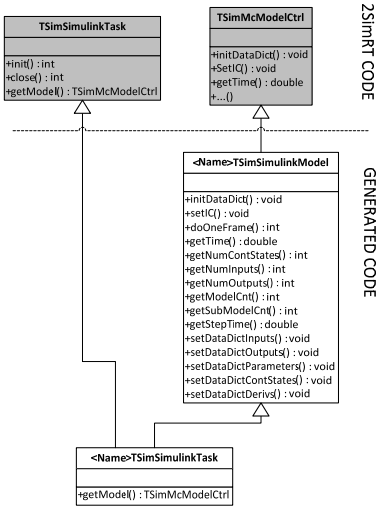


Figure 6: 2SimMC-Component Classes.

2SimRT API and glue it with generated model code (Figure 6). <Name>TSimSimulinkModel class inherits from TSimMcModelCtrl from 2SimRT API and includes <name>.h that enables it to access Simulink model code. On the other hand <Name>TSimSimulinkTask class inherits from both model class and TSimSimulinkTask from 2SimRT

API, so that one can use this class to create a schedulable 2SimRT task for the Simulink model.

TSimSimulinkModelDataDict_cpp.tlc generates a source file for setters and getter of the data dictionary. These setters and getters allow their users to access and modify model parameters, contentious states and state derivatives as well as model input and outputs.

<name>ModelDefines.h is a helper file to specify model wide global parameters. Below is an excerpt from *TSimSimulinkModeDefines_h.tlc* that demonstrates a sample scripting in a TLC file.

```
#ifndef _%<Name>TSimSimulinkModelDefines_H_
#define _%<Name>TSimSimulinkModelDefines_H_

#define MODEL %<CompiledModel.Name>
#define RT
#define NUMST %<CompiledModel.NumSynchronousSampleTimes>

#define NCSTATES %<CompiledModel.NumContStates>
```

Above one can see how TLC tokens are used to get information from the model and incorporate them in the source code. As an example, number of continuous states are defined using the token CompiledModel.NumContStates.

As a final step, Target Language Compiler is used as a model-to-text transformation tool for generation signal specifications for inputs and outputs of the model as ASCII files. With these two TLC files, *TSimSimulinkModel_Input_scd.tlc* and *TSimSimulinkModel_Output_scd.tlc,* the limits of Target Language Compiler are pushed to generate project or platform specific files. Below is an excerpt from these TLC files.

```
%with DataTypes.DataType[dataTypeIdx]
%foreach eId = numElements
    %assign eName   = Elements[eId].Name
    %assign eWidth  = Elements[eId].Width
    ...
  %%--------------------
| %<scdNr> |...| %<eName> | %<eWidth> |...| %<eName>
       %assign scdNr = scdNr + 1
  %%--------------------
%endforeach
```

5 A SAMPLE MODEL INTEGRATION

In this section, the model integration workflow that has been introduced will be used for a sample Simulink model. We will use an open source Quadrotor Flight Dynamics and Control Model (Mathworks File Exchange, 2013) (Figure 7) which implements flight dynamics and control algorithms from Bouabdallah's work (Samir, 2007).

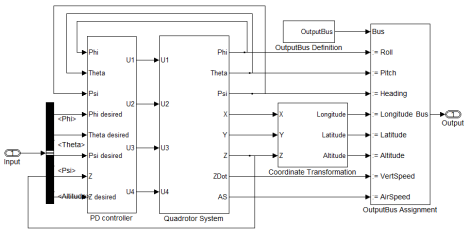


Figure 7: Simulink Model of Quadrotor Flight Dynamics and Control.

As we run *TSimModelBuilder.m* for *quadrotor.mdl*, the process will lead us through the steps of Figure 4 till to the deployment of the binaries for the model application to the specified target.

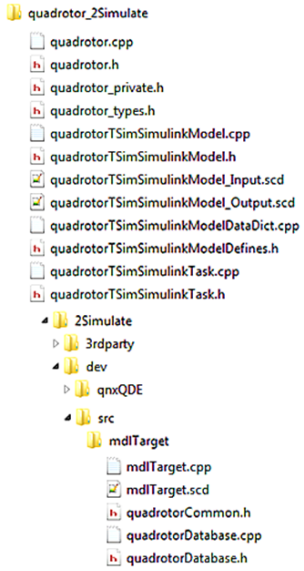


Figure 8: Generated Files.

The workflow leads to a structure that is presented in Figure 8. The files generated by the Target Language Compiler stay under the root of *quadrotor_2Simulate* directory. The Model Integration Framework that includes 3rdparty dependencies, model application source code and QNX Development Environment project is checked out to *2Simulate* directory.

The main routine of the model application code can be found in *mdlTarget.cpp*. As given in the following code excerpt, 2SimRT application

possesses three tasks. The first one is for the model, the second one is for the Control Center and the last one is for UDP communication to other simulation components.

```
//################################
// simulink model: mdlTargetmodel
//################################
quadrotorTSimSimulinkTask *pMdlTargetTSimSimulinkTask =
    new quadrotorTSimSimulinkTask ( pMdlTarget...
//################################
// CC connection
//################################
TSimConTask* pCC = pMdlTarget->addTarget(...
//#####################################
// UDP model data connection
//#####################################
TSimUdpTask *pMdlTargetIC =
new TSimUdpTask( pMdlTarget...
```

Then the generated source files are cross-compiled to QNX and the image is deployed to a QNX target using the open source tool WinSCP.

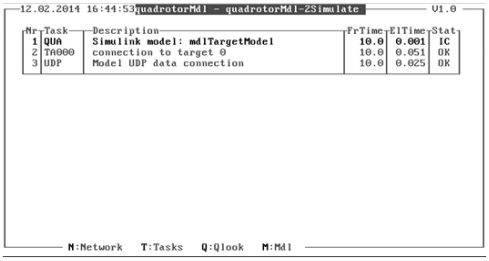


Figure 9: Console Running quadrotorMdl.

The Model Integration Workflow presented in this paper ends when the image of the model application is deployed to the specified target. The deployment scenarios launch mechanisms and network settings vary between simulators. Figure 9 presents a snapshot from a QNX console that runs the deployed model application image.

6 CONCLUSIONS

As in other research flight simulators, model update is a constant process also in DLR AVES. Flight systems researchers work for extending and enhancing their models or their systems. The presented Model Integration Workflow intends to make shortened Time-to-Deployment. Furthermore, with automated code generation and deployment process, man made errors are avoided.

With this workflow, the flight systems researchers, that use MBDSD practices, are supported for easy and fast integration and deployment of their models. Users of this workflow can integrate and deploy their models in AVES within a minimum time.

This workflow is currently operated over the Matlab command prompt. While it supports update to deployment process, it lacks in intuitive user interface for configuration and execution. Furthermore it has no support for run time monitoring and debug. Future plans include developing a graphical user interface and Simulink blocks for run time monitoring. Thus the users of the workflow will be able to monitor and debug their models that run in AVES from Simulink.

REFERENCES

Advani, S., Giovannetti, D. & Blum, M., 2002. Design of a Hexapod Motion Cueing System fir NASA Ames Vertical Motion Simulator. In *AIAA Modeling and Simulation Technologies Conference and Exhibit*. Monterey, California, 2002. AIAA.

Allerton, D.J., 1999. The Design of a Real-Time Engineering Flight Simulator for the Rapid Prototyping of Avionics Systems and Flight Control Systems. *Transactions of the Institute of Measurement and Control*, pp.51-62.

Allerton, D., 2009. *Principles of Flight Simulation*. West Sussex, United Kingdom: John Wiley & Sons, Ltd.

Duda, H., Gerlach, T., Advani, S. & Potter, M., 2013. Design of the DLR AVES Research Flight Simulator. In *AIAA Modeling and Simulation Technologies (MS) Conference*. Boston, MA, 2013. AIAA.

Estrada, R.G., Sasaki, G. & Dillaber, E., 2013. Best practices for developing DO-178 compliant software using Model-Based Design. In *AIAA Infotech@Aerospace (I@A) Conference*. Boston, MA, 2013. AIAA.

Fey, I. & Stürmer, I., 2007. Quality Assurance Methods for Model-based Development: A Survey and Assessment. In *SAE World Congress & Exhibition*. Detroit, Michigan, 2007. SAE.

Fielding, C., 2010. Model-Based Design on Flight Control Systems. In *Mathworks Model-Based Design Conference*. Daventry, UK, 2010. Mathworks, Inc.

Gotschlich, J., Gerlach, T. & Durak, U., 2014. 2Simulate: A Distributed Real-Time Simulation Framework. In *ASIM STS/GMMS Workshop 2014*. Reutlingen, Germany, 2014. ASIM.

Guido, S. & Thompson, R., 2008. Development of AUTOSAR Software Components within Model-Based Design. In *Proc. SAE World Congress & Exhibition*. Detroit, MI, 2008. SAE.

Klaes, S., 2000. ATTAS Ground Based System Simulator -An Update-. In *AIAA Modeling and Simulation Technologies Conference and Exhibit*. Denver, CO, 2000. AIAA.

Mathworks File Exchange, 2013. *PD Control Quadrotor*. [Online] Available at: http://www.mathworks.com/matlabcentral/fileexchange/41149-pd-control-quadrotor-simulink [Accessed 10 February 2014].

Miller, R., 2007. Automatic Code Generation at Nortrop Grumman. In *Mathworks Aerospace and Defence Conference*. Manhattan beach, CA, 2007. Mathworks, Inc.

Nixon, D.W., 2004. Flight Control Law Development for the F-35 Jointr Strike Fighter. In *The Mathworks International Aerospace and Defence Conference*. Newton MA, 2004. Mathworks, Inc.

Ruff, R., Stephans, C. & Mahapatra, S., 2012. Applying Model-Based Design to Large-Scale Systems Development: Modeling, Simulation, Test, & Deployment of a Multirotor Vehicle. In *AIAA Modeling and Simulation Technologies Conference*. Minneapolis, Minnesota, 2012. AIAA.

Saager, P., 1990. Real-Time Hardware-in-the-Loop Simulation for 'ATTAS' and 'ATTHeS' Advanced Technology Flight Test Vehicles. In *AGARD Guidance and Control Panel, 50th Symposium*. Izmir, Turkey, 1990. NATO.

Samir, B., 2007. *Design and Control of Quadrotors with Application to Autonomous Flying*. Ph.D. Thesis. Lausanne: École Polytechnique Fédérale de Lausanne.

Smith, R.M., 2000. A Description of the Cockpit Motion Facility and the Research Flight Deck Simulator. In *AIAA Modeling and Simulation Technologies Conference and Exhibit*. Denver, CO, 2000. AIAA.

Stroosma, O., van Paassen, R. & Mulder, M., 2003. Using the Simona Research Simulator for Human-Machine Interaction Research. Austin, Texas, 2003. AIAA.

Stürmer, I. & Pohlheim, H., 2012. Model Quality Assessment in Practice: How to Measure and Assess the Quality of Software Models During the Embedded Software Development Process. In *Int. Congress of Embedded Real Time Software and Systems (ERTS 2012)*. Toulouse, France, 2012. ERTS.

Sullivan, B.T. & Soukup, P.A., 1996. The NASA 747-400 Flight Simulator: A Natonal Reseource fir Aviation Safety Research. In *AIAA Flight Simulation Technologies Conference*. San Diego, CA, 1996. AIAA.

The MathWorks, Inc., 2007. *Matlab Product Help: Consulting Model Advisor*. Help Document. Natick, MA: The MathWorks, Inc. The MathWorks, Inc.

The Mathworks, Inc., 2014a. *Simulink Coder: Generate C and C++ Code frim Simulink and Stateflow*. [Online] Available at: http://www.mathworks.com/products/datasheets/pdf/simulink-coder.pdf [Accessed 08 April 2014].

The Mathworks, Inc., 2014b. *Simulink® Coder™ Target Language Compiler*. Help Document. Natick, MA: The MathWorks, Inc.

White, M.D. & Padfield, G.D., 2006. The Use of Flight Simulation for Research and Teaching in Acedemia. In *AIAA Atmospheric Flight Mechanics Conference and Exhibit*. Keystone, CO, 2006. AIAA.

B.6 Adapting Functional Mockup Units for HLA-Compliant Distributed Simulation

Faruk Yilmaz, Umut Durak, Koray Taylan and Halit Oguztuzun. 2014. Adapting Functional Mockup Units for HLA-compliant distributed simulation. In *Proceedings of the 10th International Modelica Conference*, Lund, Sweeden, 247-257.

Adapting Functional Mockup Units for HLA-compliant Distributed Simulation

Faruk Yılmaz, Umut Durak, Koray Taylan

Halit Oğuztüzün

Roketsan Missiles Inc.
Ankara, Turkey
[fyilmaz|udurak|ktaylan]@roketsan.com.tr

Middle East Technical University
Ankara, Turkey
oguztuzn@metu.edu.tr

Abstract

Conceptual design of systems requires aggregate level simulations of the designed system in its operational setting. By this way, performance of the system and its interactions with the other entities in its environment can be evaluated. The complex nature of these simulations often requires distributed execution. IEEE 1516 High Level Architecture (HLA) is a widely accepted standard architecture for distributed aggregate level simulations. Functional Mock-up Interface (FMI) is a recent standardization effort that leads to a tool independent systems simulation interface that enables model reuse and co-simulation. This paper aims to present a method for developing HLA-compliant federates using FMI. The method enables a Functional Mock-up Unit to join an HLA-compliant federation as a member.

Keywords: Functional Mockup Interface; High Level Architecture; Distributed Simulation

1 Introduction

Systems development process starts with conceptual design phase in which designers create concepts and conduct trade off analysis. Modeling and simulation have always been essential tools for conceptual design. Early stage systems modeling aims to identify the system requirements and its interactions with its operating environment. Effect based models, integrated in a large scale operational settings are used to evaluate the performance of the system concerning the accomplishment of its mission. Simulation of the mission space of a system requires modeling large number of entities and often simulating them in a distributed fashion. IEEE 1516 High Level Architecture (HLA) standard [1] [2] [3] is commonly used to integrate loosely coupled models of the entities in a mission space.

The Functional Mock-up Interface (FMI) is a newly developed, tool-independent model interface standard [4] [5]. Its main purpose is to model reuse between various modeling tools and environments throughout the systems development phases. A simulation component conforming to FMI is called a Functional Mock-up Unit (FMU), whose contents include a model description file, user defined libraries, source codes, model icons and documentation.

FMI and HLA has completely different behavior. While HLA supports to work at process level, the master of the FMU does not care about the topics such as entity transfer, shared resource management, time synchronization or ownership management [10].

On the other hand, there is a potential to reuse existing FMUs as federates in an HLA-compliant distributed simulation, i.e. federation. By this way, FMI will also serve as a model interface for distributed simulation entities in the concept of design phase. Here in this study, we introduce a mechanism to develop Functional Mockup Unit Federates (FMUFd) from FMUs.

1.1 Related Work

As model based development of engineering systems are getting more popular, connecting engineering models to the distributed simulation environments is also becoming an important issue of concern [6][7]. There have been some attempts for developing such tools and methodologies. Closely related to our work,

there exist two particular efforts providing a mechanism for connecting models to HLA environments.

MatlabHLA-Toolbox [8] and HLA Blockset [9] are available toolboxes to provide HLA communication feature to the Matlab. With these toolboxes, modelers can create a federate, join a federation and start publishing and subscribing entities and events. However, these solutions can only work in Matlab environment.

In [10], authors introduce an approach to run FMI Co-Simulation environment over HLA. They employ HLA RTI as a master to synchronize the simulation that is composed entirely of FMUs. They define an Object Class derived from the interface specifications of all participating federates and let each federate out of an FMU publish or subscribe its required attributes. In contrast, our approach enables participation of FMUs in a federation with non-FMU members as well.

2 Functional Mockup Interface

Functional Mockup Interface provides an interface specification for simulation components called Functional Mockup Units. FMI provides two standard interfaces, namely, FMI for Co-Simulation and FMI for Model Exchange [4] [5].

While FMI for Model Exchange specifies the interface for callers with explicit or implicit integrators, FMI for Co-Simulation specifies the interface for simulation runnables that possess solvers in them. As we can view HLA Federates as standalone simulation runnables, this effort is based on FMI-Co-Simulation interface for federate development. In this work, the first version of the standard is used as the baseline [5].

2.1 FMI for Co-simulation

As mentioned above, FMI for Co-Simulation is a standard interface for the model output containing its solver inside. Therefore, the user does not need to know which integration method is actually employed to solve the ordinary differential equations within the model.

For each of the FMU in a co-simulation environment, the communication capabilities are configured in a model specific XML file, namely *ModelDescription.xml* file. Communication with an FMU can only be realized in a discrete communication point, which is a sampling point or a synchronization point of the FMU [5].

2.1.1 Computational Flow

As show in Figure 1, FMU co-simulation computational flow has three main states, namely Instantiation and Initialization, Running and Termination.

Instantiation and Initialization

A new FMU instance is created and initiated to be ready to run. Memory allocations and initial value setting for the FMU parameters are performed in this phase.

Running

In this phase, FMU model is executed via calling *doStep()* method. Intuitively, before running a step, FMU input parameters are set by calling *FMUSetXXX(...)* and after the completion of this step the model output parameter are read by the master via calling *fmiGetXXX(...)*.

Termination

The model component is unloaded and the memory is cleaned up in this phase.

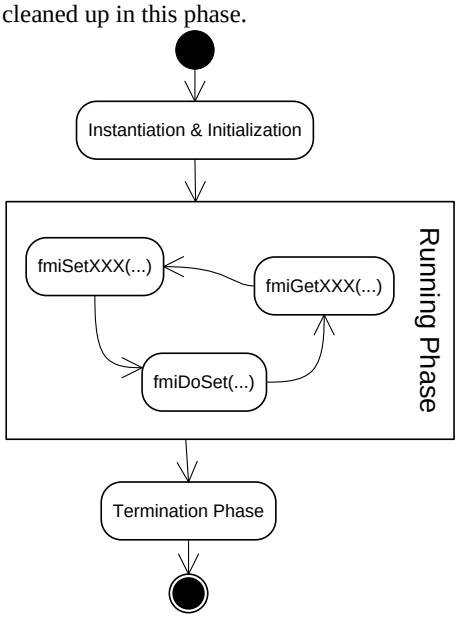


Figure 1 – FMU Co-Simulation Model Computational Flow

3 High Level Architecture

The High Level Architecture (HLA) is a common framework for distributed simulation systems. HLA promotes interoperability between heterogeneous simulations and supports the reuse of models in different contexts. HLA provides communicating data and synchronization actions between simulation members regardless of their computing platforms [1].

HLA combines simulations (federates) into a larger simulation (federation), where federates are

DOI 10.3384/ECP14096247

components and federations are component based applications. The HLA requires runtime infrastructure (RTI) software to support the operation of a federation execution. RTI provides a set of services and by using these services a federate can interact with the federation at runtime. How a federate can utilize these services is defined by the Federate Interface Specification [2].

Federation Object Model (FOM) is the HLA Federation Object Model that describes all of the object classes and interactions, attributes of object classes and parameters of interactions for the federation. Also, FOM establishes the information model contract which governs the simulation. Simulation Object Model (SOM), on the other hand, is the HLA Simulation Object Model that describes the object classes and interactions, attributes of object classes and parameters of interactions information which are exposed or consumed by a federate [2] .

3.1 HLA Services

HLA provides six groups of services to enable distributed simulation in an aggregate level [2]. Federation Management describes how to create, join, resign and manage federations, save and restore federation states. Declaration Management defines the publishing and subscribing to objects and attributes. Object Management service states how to register new instance of an object class or interaction, update the attributes, receive interactions, discover new instances and receive updates of attributes. Ownership Management defines acquisition of ownership of the registered objects. Time management describes how a federate can advance its logical time along with other federates and how to deliver the time-stamped events ensuring that a federate can never receive an event with a timestamp less than the federate's logical time. Data distribution management defines the production and consumption of data to bind the relevance of communication data among federates. As a result, RTI can recognize the irrelevant data and prevents its delivery to consumers.

3.2 HLA Object Model

HLA provides object classes and interactions as the object models, which are used to publish/subscribe the data over distributed simulation environment. Providing the data exchange between federates is one of the responsibilities of the RTI.

An **object class** can be derived from another object class. *HLAobjectRoot* is the base class of the all object classes. Each object class can contain one or more attributes. Derived classes also inherit base class attributes. Attributes are have data types. A federate will publish/subscribe only interested attributes of an object class; it does not have to deal with all the attributes in an object class.

An **interaction** can be derived from another interaction. *HLAinteractionRoot* is the base class of the all interactions. Each interaction contains one or many object parameters. Derived interactions takes base interaction parameters also. Parameters have data types. A federate should fill all the parameters of an interaction to publish it.

HLA provides six different **data types** where user can create variety of data structures by using those data types. The published/subscribed values are stored in these data structures. The details of data types are given below [3]:

- **Basic Datatype**: Basic data refers to a predefined set of data representations. Following data types should be defined by any OMT: *HLAinteger16BE, HLAinteger32BE, HLAinteger64BE, HLAfloat32BE, HLAfloat64BE, HLAoctetPairBE, HLAinteger16LE, HLAinteger32LE, HLAinteger64LE, HLAfloat32LE, HLAfloat64LE, HLAoctetPairLE, and HLAoctet*.
- **Simple Datatype**: The simple data type table refers to simple, scalar data items. Following data types should be defined by any OMT: *HLAASCIIchar*, *HLAunicodeChar*, and *HLAbyte*.
- **Enumerated Datatype**: The enumerated data type refers to data elements that can take on a finite discrete set of possible values. Following data type should be defined by any OMT: *HLAboolean*.
- **Array Datatype**: The array data type table refers to indexed homogenous collections of data types; these constructs are also known as arrays or sequences. Following data types should be defined by any OMT: *HLAASCIIstring*, *HLAunicodeString*, and *HLAopaqueData*.
- **Fixed Record Datatype**: The fixed record data type table refers to heterogeneous collections of types; these constructs are also known as records or structures. This allows users to build structures of data according to the needs of their federate or federation.
- **Variant Record Datatype:** The variant record data type table refers to discriminated unions of types; these constructs are also known as variant or choice records.

3.3 HLA Padding Rules

HLA requires that certain types of data start at a particular kind of location. Therefore, usually there is a requirement for extra bytes, namely padding bytes,

between data fields in a structure. To illustrate, consider a structure where the first field is a byte and second field is a double. Double must start at a position which is a multiple of 8. Therefore, seven bytes of padding is needed between byte field and double field for a proper structure.

The padding rules are used to determine exact positions of the fields of a data type, which constructs the data structure of an attribute.

These rules for constructed data types (arrays, fixed records, and variant records) as described below [3]:

Base Datatype

Each base type has a boundary value as provided in table 1. During the calculation of padding, this table is used to calculate structure boundary value.

Table 1 – Basic Datatype Boundary Values

Basic representation	Octet Boundary Value
HLAoctet	1
HLAoctetPairBE	2
HLAinteger16BE	2
HLAinteger32BE	4
HLAinteger64BE	8
HLAfloat32BE	4
HLAfloat64BE	8
HLAoctetPairLE	2
HLAinteger16LE	2
HLAinteger32LE	4
HLAinteger64LE	8
HLAfloat32LE	4
HLAfloat64LE	8

Simple Datatype

Same base data type padding rules also apply for simple datatype.

Enumerated Datatype

Same base data type padding rules also apply for enumerated datatype.

Fixed Record Datatype

The padding bytes are added to each field when necessary to ensure that the next field in the record is properly aligned. After a field the padding bytes can be calculated by using the following formula:

$$(Offset_i + Size_i + P_i) mod\ V_{i+1} = 0$$

where $Offset_i$ refers to the offset of the *i*'th field of the record as bytes,

$Size_i$ refers to the size of the *i*'th field of the record as bytes,

V_{i+1} is the octet boundary value of field *(i + 1)*th of the record.

Variant Record Datatype

The *HLAvariantRecord* encoding shall consist of the discriminant followed by a field. This field is chosen by using the value of discriminant. The discriminant is placed at offset 0 of the record. The padding bytes P are calculated by using the following formula:

$$(Size + P)\ mod\ V = 0$$

where $Size$ refers to the size of the discriminant as bytes, and V refers to the maximum of the octet boundary values of the alternatives.

HLA Array Datatype with Fixed Cardinality

The padding bytes P_i between *i*'th and *(i+1)*th elements can be calculated by using following formula:

$$(Size_i + P_i)\ mod\ V = 0$$

where $Size_i$ is the size of the *i*'th element of the array in bytes,

V is the octet boundary value of the element type.

HLA Array Datatype with Variant Cardinality

The first 4 bytes are used to present the number of the elements in the array. These 4 bytes are encoded as *HLAinteger32BE*. The padding bytes can be added between the inform element and the first element of the sequence. The padding bytes can be found by using following formula:

$$(4 + P)\ mod\ V = 0$$

where V is the octet boundary value of the element type.

4 Functional Mockup Unit Federate Design

The FMI for Co-Simulation standard does not provide a specification for connecting FMUs to an HLA federation, hence, FMI Co-Simulation does not have an interface ready to utilize HLA services. Moreover, there is no convenient way to convert FMU scalar variables to HLA object class attributes, because FMI Co-Simulation only supports the following primitive types: *real*, *integer*, *string*, *Boolean* and *Enumeration*. On the other hand, HLA attributes can represent any data type structure, from basic data types to the complex data type structures. Since FMI Co-Simulation scalar variables can only map to HLA basic data types, a simulation environment using complex data types cannot be directly supported by FMI Co-Simulation.

DOI 10.3384/ECP14096247

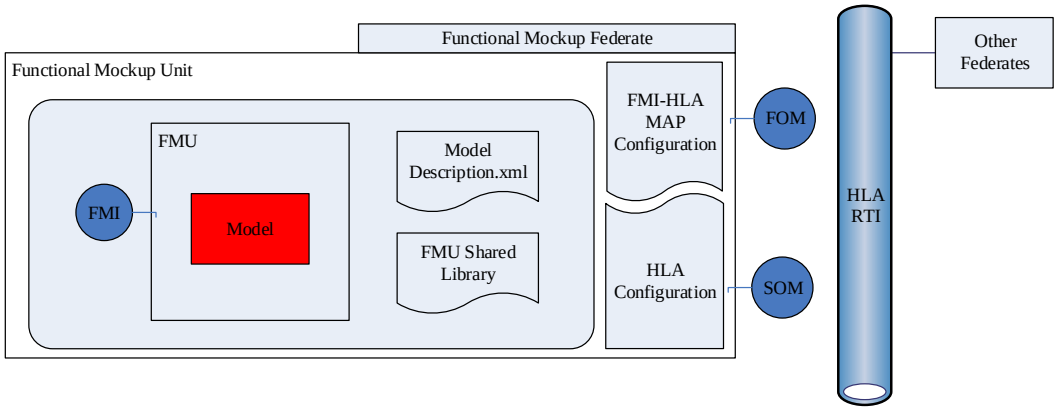


Figure 2 – Functional Mockup Unit Federate

Hence, there is a need for a wrapper that connects an FMU, in the context of FMI Co-Simulation, to the HLA distributed simulation environment. The work conducted handles the problem by designing a Functional Mockup Unit Federate (FMUFd). FMUFd has the following responsibilities:

- Instantiating, initializing, stepping and terminating of a FMU model.
- Providing the communication of distributed environment with services by using the HLA standard interface.
- Converting FMU model outputs to compatible HLA data types and sending them as HLA object updates.
- Receiving model inputs from HLA objects and converting them to compatible FMU types.

The top level structure of FMUFd that satisfies these requirements is depicted in Figure 2. The FMUFd is composed of the FMU model, FMI-HLA Map configurations and HLA connection configuration. FMI-HLA Map Configuration is used to inform FMUFd about HLA FMI relation. For each FMU, using this structure an FMUFd is needed to be configured. HLA connection configuration is related with the federation and FOM information of distributed simulation environment. By using these data, FMUFd runs with stepwise activities. As shown in Figure 3, these activities can be grouped into four main phases, namely, initialization, object discovery, stepping and termination.

In **initialization** phase, FMUFd loads and initializes the FMU and then connects the HLA federation as a federate with related HLA services and declare interested object classes for publishing and subscribing.

In **object reflection** phase, the subscribed object class instances are discovered and their values are reflected.

The **stepping** phase is the main phase of the simulation. In this phase, FMU input variables are reflected from related HLA objects, FMU runs one time step, and then, FMU output values are reflected to related HLA objects.

In **termination** phase, FMUFd terminates and unloads FMU, resigns from federation, frees allocated memory and finally stops.

The details of these steps with the process of connecting FMU to the HLA simulation environment will be described in the following sections. After that, the FMUFd capabilities in terms of the HLA services and FMUFd limitations will be mentioned briefly.

4.1 Loading an FMU

The loading of FMU takes two phases: In the first phase, the model description file is parsed, while the FMU is loaded and initialized in the second phase.

FMU Model description file provides the static information of all exposed variables and model related data. FMUFd uses model description file to identify scalar variables with data types and value reference, Globally Unique Identifier (GUID) and the model name. The scalar variables are used in data flow between FMUFd and FMU model. GUID is used for validating concrete coded FMU with model description file. Model name is used to load shared object and FMI functions. The dynamic link library has the same name as the model; shared object FMI functions should also take the model name as a prefix to their functions [5].

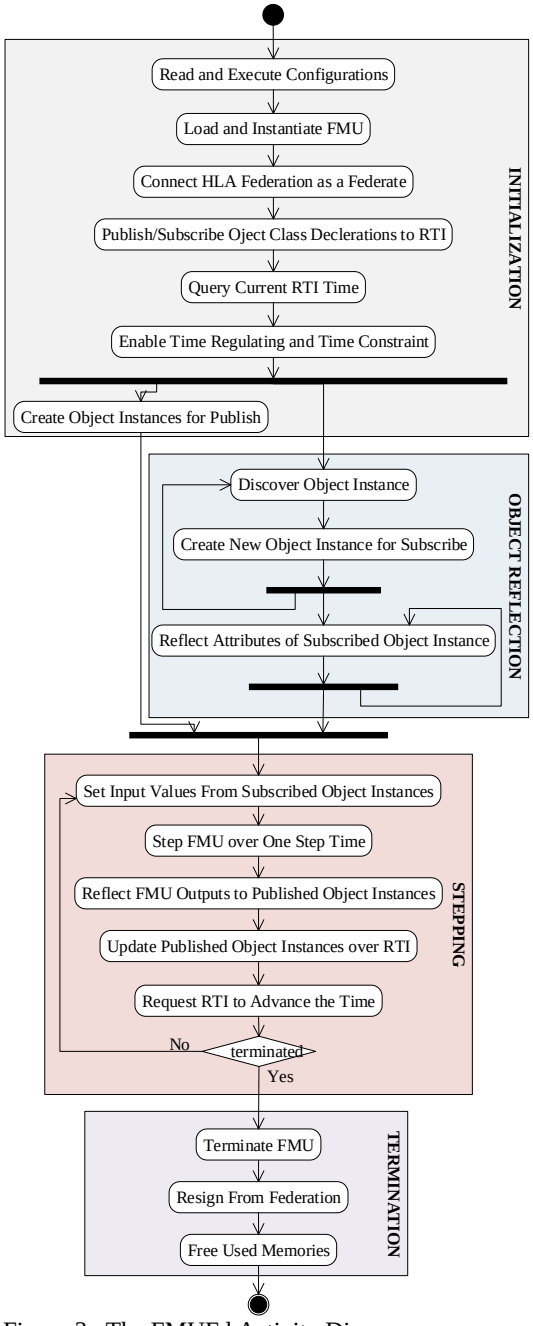


Figure 3– The FMUFd Activity Diagram

The FMU related operations are developed based on the FMU SDK [11]. By using these operations, FMUFd can load and use the FMU. The shared library, inside the FMU file should supply FMI Co-Simulation interface implementations. FMUFd loads those implementations automatically and then instantiates the model and gets the model instance. By then, FMU is ready to run steps over time.

4.2 FMU as an HLA Federate

This section describes how the FMUFd can join a federation execution as a member federate.

4.2.1 Connect to the HLA Federation

The FOM file contains all data exchange related information of HLA Federation, including object classes and object class attributes. By using this file, the FMUFd identifies the structure of each object class with attributes and data types.

The parsing process of a FOM takes two steps. First of all, the data types are parsed and stored in a map. For each type of the data, different parsing procedure is applied as each type has its special fields. For example, the size and endian information is set only for basic data types. Then, the object classes are parsed with their attributes and the data type of each attribute is retrieved from the map. If a class is derived, then its inherited attributes are obtained from its ancestors.

After parsing the FOM file, FMUFd tries to connect to the federation. The federation information is provided by the user through a configuration file. The FMUFd reads this file to get the federation name, path of FOM file and the name of its own federate. Then, FMUFd tries to create a federation if it has not been created yet. Finally, it joins the federation.

After joining the federation, FMUFd declares RTI which object classes with which attributes will be published and/or subscribed. This information is provided by the user with a SOM file. The FMUFd reads the file and identifies the published/subscribed objects and informs the RTI.

4.2.2 Create Object Instances

There are two scenarios for creating the object instances. At the beginning of the simulation, after declaring the object classes, the FMUFd creates the object class instances for publishing the FMU output parameters. The initial values of this object can be assigned by user with using the configuration files. Then, whenever an object class is discovered (new object instance is subscribed), the FMUFd creates an instance of the discovered object class.

Each object contains both object class metadata and attributes. Each attribute allocates the memory with the same size as its data type. While calculating the size of a data type the padding rules are used as described in section 3.3. Although there exists some

DOI 10.3384/ECP14096247

rules, still it may not be straightforward to find the exact size of the data type. For example, fixed record data type can contain another fixed record data type and a dynamic array data type. In this case, it is not possible to find exact size of the data type without filling the exact data. Therefore, for every update, the size of the data type should be recalculated. This recalculation may have a problem regarding the performance of an application. To address this issue, the FMUFd has been designed with two restrictions:

- The array data type with dynamic cardinality is not supported by FMUFd,
- The discriminant value of the variant record data type is explicitly defined in configuration file and cannot be changed in runtime.

With these restrictions, the FMUFd calculates the size of each attribute at the beginning of the simulation and uses this size throughout the simulation. As the data can contain different data types in it, the calculation may be performed through recursion. The basic data type is the only type with known size. Whenever the recursion reaches a basic data type, the padding rules are applied. The base case of the recursion could be the code segment given in Figure 4. The *currentOffset* value is passed into recursion which holds the previously calculated offset. After recursion is finished, *currentOffset* will hold the size of the root data type.

```
if(theDataType->type ==
ObjectClass::Attribute::DataType::BasicData)
{
  int mod = theDataType->size;
  int padding = (theDataType->size
                              - (currentOffset % mod))
                              % mod;
  theDataType ->offset = currentOffset + padding;
  currentOffset = newDataType->size
                              + theDataType->offset;
}
```

Figure 4 – The base condition code snapshot for calculating the padding bytes

4.2.3 Update/Reflect Object Class Attribute

A complex data type can contain both big endian and little endian data types in it, independent from application computer's endian type. Therefore, before updating the object class attribute, the attribute values should be encoded to the right type of endian. Likewise, after reflecting the attribute, the value should be decoded to the computer endian type. The FMUFd always keeps the data with the same encoding of computer. By doing that, it becomes easier to use the data in an application. Whenever an attribute is needed to be updated, the attribute is encoded first then update operation is called. Likewise, whenever an attribute is reflected, the value of that attribute is decoded first and kept in decoded form in memory.

The encode/decode operation is also executed with recursion. The basic data type is the only type with known endian type. Whenever the recursion reaches to the basic data type, the swapping operation is applied. The base case of the recursion could be the code segment given in Figure 5. The *returnValue* and *rowData* are the void* data type values, with the same size of attribute. If the recursion is used for updating the attribute operation than *rowData* refers to the current value of the attribute, otherwise, it refers to the reflected value of the attribute. The *returnValue* refers to the encoded (or decoded) value of the attribute.

```
if(dataType.type
== OjectClass::Attribute
::DataType::DataTypeType::BasicData)
{
  if(currentNodeEndianType
    != dataType.endianType )
  {
    T_UINT8* returnValueOffset
      = (((T_UINT8*) returnValue )
        + dataType.offset);
    T_UINT8* rowDataOffset
      = (((T_UINT8*) returnValue )
      + dataType.offset);

    switch(dataType.size)
    {
    case sizeof(T_UINT8):
      *returnValueOffset = *rowDataOffset;
      break;
    case sizeof(T_UINT16):
      *((T_UINT16 *) returnValueOffset)
= _byteswap_ushort(*(T_UINT16 *) rowDataOffset);
      break;
    case sizeof(T_UINT32):
      *((T_UINT16 *) returnValueOffset)
= _byteswap_ulong(*(T_UINT16 *) rowDataOffset);
      break;
    case sizeof(T_UINT64):
      *((T_UINT16 *) returnValueOffset)
= _byteswap_uint64(*(T_UINT16 *) rowDataOffset);
      break;
    }
  }
```

Figure 5 – The base condition code snapshot for encoding/decoding the attribute values.

4.3 Running the Federate

After introducing how HLA data is de-marshalled, the next step is mapping FMU scalar variables to HLA basic data types. This mapping is performed through user configuration files. These files inform the FMUFd about which data from HLA will be set to FMU and which data from FMU will be published to HLA.

After mapping between FMU scalar variables and the HLA attributes, the stepping function can be executed.

Before running a step of FMU, the FMUFd updates each input variable of the model. The input values are obtained from an instance of related object class. If there is no instance for related object class then the FMUFd will wait for the instance of that related object.

After running a step of FMU, the FMUFd updates related attributes of the HLA objects by retrieving the values from related FMU scalar variables. Therefore, an FMU output values can be mapped to different HLA objects, which are controlled by the FMUFd. After value updates are finished, the FMUFd will request the RTI to publish those attributes.

4.4 FMUFd Structure

The FMUFd is implemented as an application. Inspired from paper [14], the application is constructed with three base layers as Figure 6.

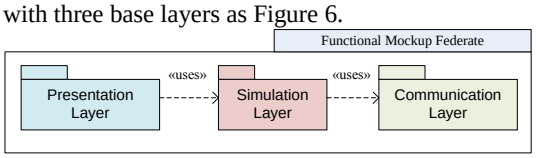


Figure 6 – the FMUFd Structure

4.4.1 Presentation Layer

The presentation layer is the user interface of the application. This layer provides presentation of application, input and interaction with the user as shown in Figure 7. The plot in the figure shows the change of some parameters of the missile and target over time. By using this layer, a user can load the necessary configurations to FMUFd and observe scalar variables' value changes in real time.

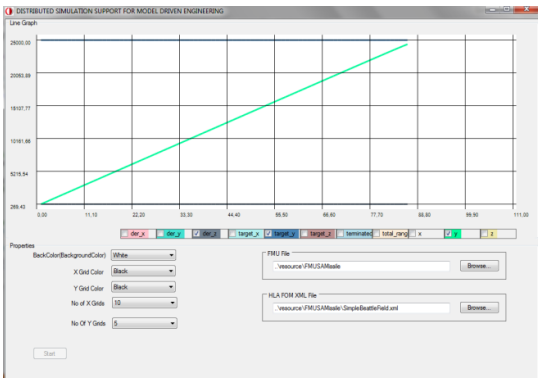


Figure 7 – The FMUFd screenshot while running the missile FMU

4.4.2 Simulation Layer

The simulation layer processes the application. It includes the computation of FMI simulation and federate specific HLA object classes. Its purpose is to run FMU and generate the federate behavior.
Simulation layer is responsible for running the simulation. This layer initializes the FMU, supplies necessary inputs for FMU from HLA class instances, runs the models and publishes the model outputs over HLA distributed environment.

One of the key features of the simulation layer is to create HLA object class structure dynamically. That is, without having the real structure, simulation layer can create a void data with the same size of the structure by using FOM xml file. Then simulation layer can edit this void data parts with the same position of any object class attribute fields.

4.4.3 Communication Layer

The communication layer deals with the RTI communication in order to access the object classes and interactions exchanged in the federation execution. RTI is the middleware that manages the federation execution and object exchange through a federation execution. In addition to data exchange, communication layer also supports time management service.

4.5 FMUFd Capabilities

In this section, FMUFd capabilities are described in terms of HLA interface services. Data distribution management and ownership management are not used in our current implementation.

Federation Management: If the federation has not been created before, the FMUFd creates the federation. Then it joins the federation. Similarly, after simulation is finished, the FMUFd resigns from the federation and if there is no other federate connected to the federation, it destroys the federation.

Declaration Management: FMUFd informs the RTI about publishing/subscribing object classes with attributes.

Object Management: Whenever new object class instance is discovered, FMUFd keeps the handle for this instance and allocates memory for it. Whenever a *reflectAttributeValues* event is raised by RTI, the FMUFd check whether the object instance is discovered before. If it is discovered, FMUFd reflects the attribute values to the allocated memories of the object instance, and ignores otherwise. Whenever a *removeObjectInstance* event is raised by RTI, the FMUFd checks if the object instance is discovered before. If it is discovered, FMUFd deletes the handle of instance and frees the related allocated memory.

DOI 10.3384/ECP14096247

FMUFd reflects the attribute values to the allocated memory of the object instance, ignores otherwise.

Time Management: FMUFd works as a time regulating and time constraining federate. As the nature of the time constraint, FMUFd ensures that the subscribed object model instance received reflection no less than the *currentRTITime.* Also, after each running step of the model, FMUFd requests to update the federate time.

4.6 Limitations

FMUFd still has some limitations and constraints. First of all, HLA interactions are not supported by FMUFd as there exists no corresponding logical concept in current FMI for Co-Simulation standard. Moreover, the array data type with dynamic cardinality is not supported by FMUFd. Finally, the discriminant value of the variant record data type is defined at the beginning of the simulation and FMUFd does not allow changing it at runtime.

5 Demonstration

To demonstrate the FMUFd usage, a simple distributed simulation environment is developed with MAK HLA RTI [12] implementation. For this application, the RPR2-D17 FOM file developed by SISO [13] is used as a FOM file. The used HLA object classes with its parent hierarchy are shown in Figure 8.

```
HLAobjectRoot.BaseEntity.PhysicalEntity.Platform.Aircraft
HLAobjectRoot.BaseEntity.PhysicalEntity.Munition
```

Figure 8 – RPR2-D17 FOM classes used in the case study.

There are three nodes connected over an Ethernet network in this distributed simulation environment as shown in Figure 9. In the missile node, the missile co-simulation FMU (called *MissileFMUFd*) is connected to distributed simulation environment as the HLA federate by using FMUFd. Similar to missile PC, the aircraft co-simulation FMU is also connected to the simulation environment as a federate by using FMUFd (called *AircraftFMUFd*) in the target aircraft PC. The synthetic environment node is used to provide other entities in this operational setting, such as the missile launch platform, and to visualize the simulation in 2D and 3D. To this end, Presagis STAGE is used [15].

With two configuration files, one for FMU inputs and one for FMU outputs, user should supply information to the FMUFd about mapping between FMU scalar variables and HLA basic data types. Therefore, the entire hierarchy down to the basic data types should be explicitly defined for an object model.

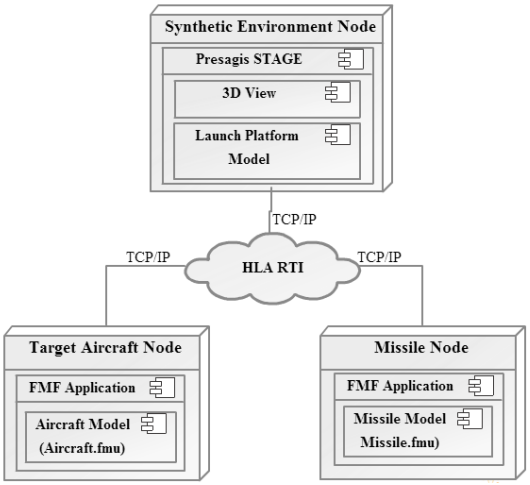


Figure 9 – The Deployment View Diagram of Simulation Environment

The example extract from a configuration file is provided in Figure 10. This example shows how the *Target_Ecef_X* scalar variable can be mapped with the *Aircraft* object's *Spatial* attribute's data type where data type goes down the hierarchy until it reaches the basic data type *HLAfloat64BE*. This mapping is specified for other scalar variables as well.

```
Target_Ecef_X = Aicraft|Spatial|SpatialStruct
:DeadReckoningAlgorithm-A-Alternatives
:SpatialStruct-DeadReckoningAlgorithm[DRM_FPW]
:SpatialFPStruct:WorldLocation:WorldLocationStruct:X
:HLAfloat64BEmetersperfectalways:HLAfloat64BE
```

Figure 10 – The example mapping between FMU scalar variables and HLA object class attribute data types.

With these configurations, 1500 simulation runs have been executed and performance figures for framework overhead have been measured. The median time for updating FMU parameters from HLA objects for *MissileFMUFd* is 254 microseconds. Likewise, the median time for updating HLA attributes from FMU parameters for *MissileFMUFd* is 356 microseconds. Measurement were taken on a computer with Intel Xeon 2.66GHz processor, 4GB DDR3 RAM and Windows 7 Pro 64bit operation system.

6 Conclusion

The FMI is an emerging standard for co-simulation and model exchange in Model Based Integration community. Also, HLA is a well-accepted standard for the distributed simulation. FMUFd supplies a solution for participation of FMUs that implement FMI Co-Simulation interface in an HLA Federation. Thus, a system model can be simulated as a part of an aggregate simulation of its operational setting. Moreover, this promotes a high level of reusability of system models supporting FMI.

As an alternative approach, the wrapping process may generate an FMU HLA wrapper code automatically by reading the FMU specification and generating the wrapper that knows how to translate just the specific FMU join HLA. This approach, on the other hand, is both FMU and federation specific and requires a recompile for each FMU. In our case, we aimed at recompilation free integration of FMUs to any federation via configuration files.

Finally, FMUFd is currently released in Roketsan Inc. as an in-house developed simulation infrastructure and being employed in some system development projects.

Acknowledgments

This work is supported by Turkish Ministry of National Defense, Undersecretariat for Defense Industries [Project Name: MOKA].

References

[1] HLA Working Group of IEEE: IEEE Std 1516-2000, IEEE Standard for Modeling and Simulation (M&S) High Level Architecture (HLA) - Framework and Rules. New York, USA, 2000.

[2] HLA Working Group of IEEE: IEEE Std 1516.1-2000, IEEE Standard for Modeling and Simulation (M&S) High Level Architecture (HLA) - Federate Interface Specification. New York, USA, 2000.

[3] HLA Working Group of IEEE: IEEE Std 1516.2-2000, IEEE Standard for Modeling and Simulation (M&S) High Level Architecture (HLA) - Object Model Template (OMT) Specification. New York, USA, 2000.

[4] MODELISAR Consortium: Functional Mock-up Interface for Model Exchange Version 1.0, www.functional-mockupinterface.org, January, 2010

[5] MODELISAR Consortium: Functional Mock-up Interface for Co-Simulation Version 1.0, www.functional-mockupinterface.org, October, 2010

[6] Stenzel, C.: Distributed Simulation in Technical Applications, X International PhD Workshop, OWD 2008, Conference Archives PETiS, Vol. 25, 2008, Gliwice, Poland, 513-518

[7] Lasnier, G., Cardoso, J., Siron, P., Pagetti, C. and Derler, P.: Distributed Simulation of Heterogeneous and Real-time Systems. (2013) In: 17th IEEE/ACM International Symposium on Distributed Simulation and Real Time Applications - IEEE/ACM DS-RT 2013, 30 October 2013 - 01 November 2013 (Delft, Netherlands).

[8] Stenzel, C., Pawletta, S.: CERTI - Bindings to Matlab and Fortran, University of Wismar, http://www.mb.hs-wismar.de/~stenzel/software/MatlabHLA.html Germany, 2008 (Last Accessed 20/11/2013)

[9] HLA Toolbox ™ The MATLAB® interface to HLA, http://www.forwardsim.com

[10] Awais, M. U., Palensky, P., Elsheikh, A., Widl, E. and Matthias, S.: The High Level Architecture RTI as a master to the Functional Mock-up Interface components. Vienna, AUSTRIA, 2013

[11] QTronic Company Developer Team: FMU SDK version 1.0.2, http://www.qtronic.de/en/fmusdk.html, 2010

[12] MÄK Technologies: MAK RTI User's Guide, 2013

[13] Shanks, G.: Real-time Platform Reference Federation Object Model (RPR FOM) Version 2.0D17, Simulation Interoperability Standards Organization, 2003

[14] Topçu, O. and Oğuztüzün, H.: Layered simulation architecture: A practical approach", Simulation Modelling Practice and Theory (SIMPAT) Journal, vol. 32, March 2013, pp. 1-14.

[15] Stage Sales Team: Stage, A Complete Simulation Development Environment, http://www.presagis.com, Montreal, CANADA, 2013

B.7 Ontology for Objective Flight Simulator Fidelity Evaluation

Umut Durak, Artur Schmidt and Thorsten Pawletta. 2014. Ontology for objective flight simulator fidelity evaluation. *SNE Simulation Notes Europe* 24, 2, 69-78.

SNE Technical Note

Ontology for Objective Flight Simulator Fidelity Evaluation

Umut Durak[1*], Artur Schmidt[2], Thorsten Pawletta[2]

[1] German Aerospace Center (DLR), Institute of Flight Systems, Lilienthalplatz 7, 38100 Braunschweig, Germany; *umut.durak@dlr.de

[2] University of Wismar, FIW / MVU, FG CEA, PF 1210, 23952, Wismar, Germany

Simulation Notes Europe SNE 24(2), 2014, 69 - 78
DOI: 10.11128/sne.24.tn.102242
Submitted: Sept. 15, 2014 (selected ASIM SST Post-Conf. Publ.);
Revised: Oct. 20, 2014; Accepted: October 30, 2014;

Abstract. The term simulator fidelity has become enormously important in the scope of simulation research, when assessing training efficiency and the transfer of training to real flight. It is defined as the degree to which a flight simulator matches the characteristics of the real aircraft. Objective simulator fidelity provides an engineering standard, by attacking the fidelity problem with comparison of simulator and the actual flight over some quantitative measures. Research flight simulators encompass some differences from commercial flight simulators. They require high flexibility and versatility concerning the cockpit layout and visual and motion systems, as well as flight simulation models. It should be easy to modify the flight simulation model or other software and hardware components of the simulator. To support this, there is a need for a flexible automated test methodology, in order to determine the fidelity of the most relevant simulator subsystems, since they are often modified during the life cycle of the simulator. This methodology not only shall support automated execution but also enable automated generation of the test cases which are subject to change as well as simulator components. The Institute of Flight Systems (FT) at the German Aerospace Center (DLR) has a reconfigurable flight simulator, the Air Vehicle Simulator (AVES), for research of rotorcraft and fixed-wing aircraft.
The study reported in this paper adopts a Model Based Testing approach to tackle the high flexibility requirement of AVES. The outcome of the paper is a metamodel for model-based objective flight simulator evaluation. Metamodeling has been carried out in two levels. An Experimental Frame Ontology (EFO) has been developed adopting experimental frames from Discrete Event System Specification (DEVS), and as an upper ontology to specify a formal structure for a simulation test. Then in Objective Fidelity Evaluation Ontology (OFEO) that builds upon EFO, domain specific meta-test definitions are captured.

Introduction

Since the late 1920s, when Edward Link built the 'Blue Box' [1], flight simulators have been important elements of aviation. Flight simulators became well accepted as training aids by many aircraft operators before the digital era. Highly sophisticated flight simulators have been employed commercially within civil and military flight training organizations in order to enhance pilot skills.

In the 1980s, the aeronautics research community started using flight simulators for developing and experimenting advanced concepts and conducting aviation human factors research. Some of the first examples of research flight simulators include ATTAS Ground-Based Simulator from German Aerospace Center (DLR) [2] [3], National Aerospace Agency (NASA) Crew Vehicle Systems Research Facility in Ames Research Center [4] and Visual Motion Simulation and Cockpit Motion Facility at the Langley Research Center [5]. Some more recent examples are the Air Vehicle Simulator (AVES) of DLR [6], HELIFLIGHT at the University of Liverpool [7], NASA Ames Vertical Motion Simulator (VMS) [8] and International Research Institute for Simulation, Motion and Navigation (SIMONA) of Delft University of Technology [9].

The authors define fidelity in flight simulation as the degree to which a flight simulator matches the characteristics of the real aircraft. As its effect on training efficiency and transfer of training to real flight became better understood, fidelity became a more important research subject [10]. Objective simulator fidelity assessment provides an engineering standard to qualify the degree of fidelity through objective measures. It approaches the fidelity problem with comparison of simulator and the actual flight over some quantitative cues.

Requirements for research flight simulators encompass some differences from commercial flight simulators.

They require high flexibility and versatility concerning the cockpit layout and visual and motion systems, as well as flight simulation models. They must allow easy modification of the flight simulation model or other software and hardware components of the simulator. In order to efficiently determine the fidelity of subsystems that are often modified during the life cycle of the simulator, there is a need for a flexible automated test methodology.

This methodology is required to automate not only the execution, but also the test case generation. While there are standard sets of test cases for objective flight simulator evaluation, each modification of simulator components asks for either a different subset of a standard test set or modifications in standard test specifications. Therefore, test cases are also required to be easily modifiable, as well as the components of a research simulator.

Automated testing can be applied through the use of software to control the execution of tests and a comparison of actual outcomes to the predicted ones. Available test data taken from aircraft are used as input signals to the simulator and the output signals of the simulator are compared to the measurements to be presented for the evaluator in a smart format. Braun and Galloway [11] reported their automated fidelity test system that compares directly the flight test results and manual execution of flight tests in simulators.

Wang et al. [12] [13] presented *Automated Test System* (ATS) that measure force function, evaluation function and transport delay with its non-intrusive interface with operator station. Jarvis et al. [14] summarizes the efforts on validation of sensory cues, motion cues, vibration and sound cues, visual cues, transport delays and flight dynamics models in flight simulators.

Previous efforts regarding automated testing for objective flight simulator evaluation utilized fixed test descriptions. The presented automated testing infrastructures contributed flawless execution of the tests. But they did not attack automation of test case generation. The bridge between the state of the art Model Based Testing (MBT) practices and automated flight simulator testing is still missing. MBT can be introduced as the idea of automating test case generation from a test model rather than implementing test cases manually [15].

Thus, the test case generation is made more flexible. Metamodeling is employed to capture the domain specific concepts and constraints for building test models.

Then test modeling is used to specify test cases, and these test models are translated automatically to executable test cases [16]. DLR intends to adopt an MBT approach in flight simulator domain and hereby provide a methodology for flexible automated test case generation. Therefore a metamodel is required for objective flight simulator fidelity evaluation.

A metamodel is defined as an explicit model of constructs and rules that are used to define a model [17]. Following Gruber [18], definition of ontology is "explicit specification of shared conceptualization". Moreover, metamodels are categorized as ontologies that are used by modelers [17].

Here, the test case can be defined as a sequence of input stimuli that will be fed to the System Under Test (SUT), namely test inputs and the expected behavior of the system, namely test oracle (**Figure 1**) [19].

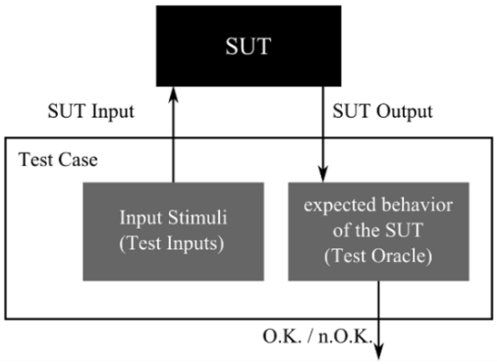


Figure 1. Test Case Structure.

Moser et al. [20] stressed that ontologies as machine-readable domain knowledge, which can be utilized for test case generation. Then Nguyen et al. [21] presented a framework for ontology driven test case generation in the context of multi-agent systems. Adopting these ideas, ontologies are employed to structure meta-test definitions.

The domain knowledge about the objective validation of simulator systems including the rules for assessing the results of test runs is captured in ontologies.

Zeigler and his colleagues developed the concept of *Experimental Frame* (EF) [22] [23]. An EF defines the conditions under which a model is to be examined. It comprises of an input generator, a verifier for the desired conditions and an analyzer for the outputs.

Following Zeigler et al. [23], the EF is critical for evaluating the model validity. Traoure and Muzy in [24] and Foures et al. in [25] published the usage of the EF approach for specifying invariant validation experiments.

In this research, metamodeling has been carried out on two levels. An *Experimental Frame Ontology* (EFO) has been developed as an upper ontology to specify a formal structure for generic simulation test model. Then in *Objective Fidelity Evaluation Ontology* (OFEO) that builds upon EFO, domain specific meta test definitions are captured. Protégé [26] is used as the ontology development environment and ontologies are developed using *Ontology Web Language* (OWL).

This paper will present these ontologies after introducing a background on objective fidelity evaluation, experimental frames and ontologies in general.

In this paper, first a background will be introduced on objective fidelity evaluation, EF and ontologies. Then EFO and OFEO will be presented. The paper will end with concluding remarks.

1 Background

1.1 Objective fidelity evaluation

Fidelity is regarded as a multivariate construct with no consensus among researchers on a single index of measurement or definition and it is strongly related to the training task to be performed with the simulator.

There are two approaches to measure simulator fidelity; the subjective and objective approaches [12]. The subjective approach tries to identify the degree of realism felt by the user. User feedback is usually collected using subjective rating scales [27].

Although subjective scales are valuable, it is hard to generalize across scales because of the individual opinions and bias of those providing assessments [12]. Objective approaches attack the fidelity problem with of simulator and the actual flight over some quantitative cues.

'ICAO 9625 Manual of Criteria for the Qualification of Flight Training Devices' [28] is the well accepted global standard for qualification of flight training devices. The standard specifies seven types of fidelity that correspond to a capability level to provide a certain type of training.

For example, simulators classed as 'Type 1' can be used for all training tasks used during completion of Private Pilot License (PPL) training, whereas 'Type 7' is required for some of the training tasks used when awarding 'Type Rating'. Appendix B of the standard specifies the test cases for objective validation of simulators. These test cases include comparison of results from tests conducted in the simulator and aircraft validation data.

The Royal Aeronautical Society (RAeS) published 'Aeroplane Simulation Training Device Evaluation Handbook Vol. 1 Objective Testing' [29] to ease the implementation and enhance the understanding of objective tests introduced in ICAO 9625. It provides further discussions about the implementation of each test and introduces some example cases with some plots. ICAO 9625 provides tables that specify each test case with parameters, tolerances and flight conditions. Table 1 shows an example test specification from the standard, for testing the minimum radius.

Test	Tolerance	Type						
		1	2	3	4	5	6	7
Minimum radius turn	±0,9m (3ft) or ±20% of aeroplane turn radius					✓		✓

Table 1: Sample Test Specification from ICAO 9625 [28]

This effort takes ICAO 9625 as a baseline to define test cases as they present a shared understanding of experts in the field. Tests are grouped under performance, handling qualities, motion system, visual system and sound system. Among these tests, those regarding performance and handling qualities are related to flight dynamics models, and have no other subsystem or device dependencies. For this reason, they are considered to better suit automation. Therefore, as a first step, the current research addresses these groups.

The RAeS introduces the benefits of employing automatic testing in objective fidelity evaluation as repeatability, ease and rapidity of conducting tests. The RAeS handbook [29] specifies the features of an automatic testing system as initializing the simulator with the test initial conditions, trimming the aircraft, creating the stimulus if required, using flight controls and finally checking the simulator output against test criteria.

1.2 Experimental frame approach

The EF approach was originally introduced by Zeigler in [22] in context with the *Discrete Event System Specification* (DEVS). The objective is the explicit separation between the model and the experiment. Moreover, an EF specifies a limited set of circumstances under which a model is to be observed. Currently, the EF approach belongs to the state of the art and it is used in many modelling and simulation projects including validation experiments [24] [25] [30] [31]. Following Zeigler [22], the formal specification of the EF is given by the 7-tuple:

$$EF = < T, I, O, C, \Omega i, \Omega c, SU >$$

where:

T is the time base
I is the set of input variables
O is the set of output variables
C is the set of control variables
Ωi is the set of admissible input segments
Ωc is the set of admissible control segment
SU is a set of summary mappings

The EF can be implemented in various ways. Zeigler [22] recommends implementing the EF as a coupled system consisting of a generator, acceptor and a transducer that is connected to a SUT. In our context, the SUT is always a model. For this reason, it is called *Model Under Test* (MUT). Figure 2 illustrates such a realization of EF coupled to a MUT schematically.

Test inputs are produced by a generator. The set of admissible input segments influences MUT's behavior. The acceptor and transducer form the test oracle. Based on output variables, the transducer calculates outcome measures in the form of performance indices, comparative values, statistics etc.

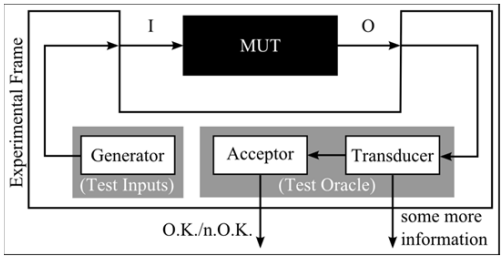


Figure 2: Illustration of EF with MUT.

The acceptor corresponds to a decision unit that decides if an experiment is valid or not. For this purpose, the acceptor monitors its inputs and maps them to a specified admissible control segment. In case of violation of the admissible control segment the experiment will not be accepted. Beside control variables, the input of an acceptor can be output variables or outcome measures.

The EF approach defines a uniform structure for a systematic experiment specification. The specification has to be coded in the description of an EF. This means that each kind of experiment needs the definition of a distinct EF.

1.3 Ontologies

Knowledge in a domain is formalized using concepts, relations, functions, axioms and instances in an ontology. Concepts can be anything about which something is said, and therefore, can be a description of a task, function, action, strategy etc. Taxonomies are widely used to organize the ontological knowledge in domain using generalization/specialization relationship through simple/multiple inheritance.

Relationships represent a type of interaction between the concepts of the domain and functions can be regarded as a special kind of relation. Axioms on the other hand are used to model sentences that are always true. They are added to ontology for several purposes, such as constraining the information contained in the ontology, verifying its correctness or deducting new information. Instances are the terms that are used to represent the elements of the domain. They actually represent the elements of the concepts [32].

Ontologies in engineering domain have been developed for various purposes including specifying engineering information systems, integration of engineering applications, supporting engineering design and development. The first efforts on developing engineering ontologies were in the 1990's. The 'PhysSys' [33] was one of the first engineering ontologies based upon system dynamics theory that is practiced in engineering modeling, simulation and design. The PhysSys was developed to formally define how design engineers or the end users of *Computer Aided Engineering* (CAE) systems understand their domain and to provide a foundation for the conceptual schema for data structuring in engineering databases, libraries and other CAE information systems [33] [34].

The ideas formalized in PhysSys provided a base for the development of a library of reusable models for engineering and design.

Fishwick and Miller in [35] discussed the venues of ontology use in modeling and simulation. One of the late examples of ontology use in modeling and simulation is reported by Durak et al. [36] [37]. The group enabled simulation reuse over an ontology driven methodology.

Another ontology-based modeling and simulation approach was established by Zeigler with the System Entity Structure and Model Base (SES/MB) framework [22] [23] [38] [39].

Today the SES is an ontology framework for conceptual system modeling and for specification of a set of modular hierarchical system structures and parameter settings.

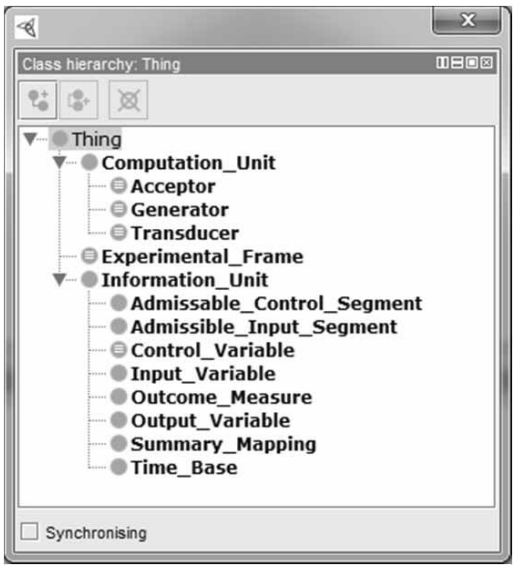


Figure 3: Entity Hierarchy of the Experimental Frame Ontology.

2 Experimental Frame Ontology

The EFO forms the upper level of the metamodel for objective flight simulation evaluation. The previously introduced EF approach is used to specify a formal structure of generic test cases. Hence, every test case has to be specified according to the EF definition in the Section 1.2.

Figure 3 illustrates the entity hierarchy of the EFO in Protégé. The first layer consists of three entities: Computational Unit, Informational Unit and the EF. Computational Units comprises the generic Acceptor, Transducer and Generator which will be presented as executable blocks in a test case. The Information Unit defines basic entities of an EF. The Experimental Frame entity thus conforms to the actual EF.

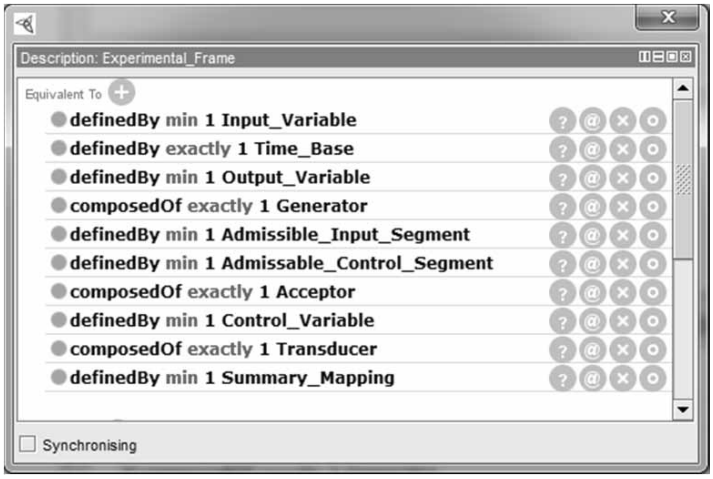


Figure 4: Description of a Generic Experimental Frame.

Furthermore particular properties are implemented to define the relations between the entities. For example the properties *composedOf* and *definedBy* makes clear that any EF is a composition of Computational Unit and is defined by the Informational Units.

As a result we obtain a generic EF which conforms to a generic test case. Thus, any test case will have the unique structure as shown in Figure 4 on its top level. The EFO forms the basis for the OFEO that will define test cases in detail.

3 Objective Fidelity Evaluation Ontology

OFEO is constructed by extending the upper level EFO that specifies any test case that will be applied to MUT using experimental frames formalism. The hierarchy of OFEO using Protégé is depicted in Figure 5. The elements from EFO can be traced in this hierarchy.

Each objective validation test case described in ICAO 9625 under performance and handling qualities are specified by an experimental frame. Thus, each test possesses a Generator, Transducer and an Acceptor. The specification of these three entities will inherently describe how this specific test will be exercised. These three entities will constitute the automatic test system.

Following the features of automated test systems introduced in the RAeS Handbook [29], the Generator is described as the component to initialize the test with initial conditions and trim the aircraft and create the stimulus following the ones from the flight test using the flight controls.

Hence, the Generator is interpreted as test independent. On the other hand, the Transducer is described as the component that will compute Outcome Measures that are required for the Acceptor for a specific test.

As an example, the Minimum Turn Radius test requires a Simulated Turn Radius to be computed from a simulation output. Or likewise, Rate of Turn versus Nosewheel Steering Angle test requires Simulated Turn Rate value to be computed.

So, a specific Transducer is defined for every test. Lastly, the Acceptor is described as the component that checking the MUT against test criteria. Since every test has a particular criterion, an Acceptor is defined for each test. Accordingly, we are expecting to have particular Control Variables for each test.

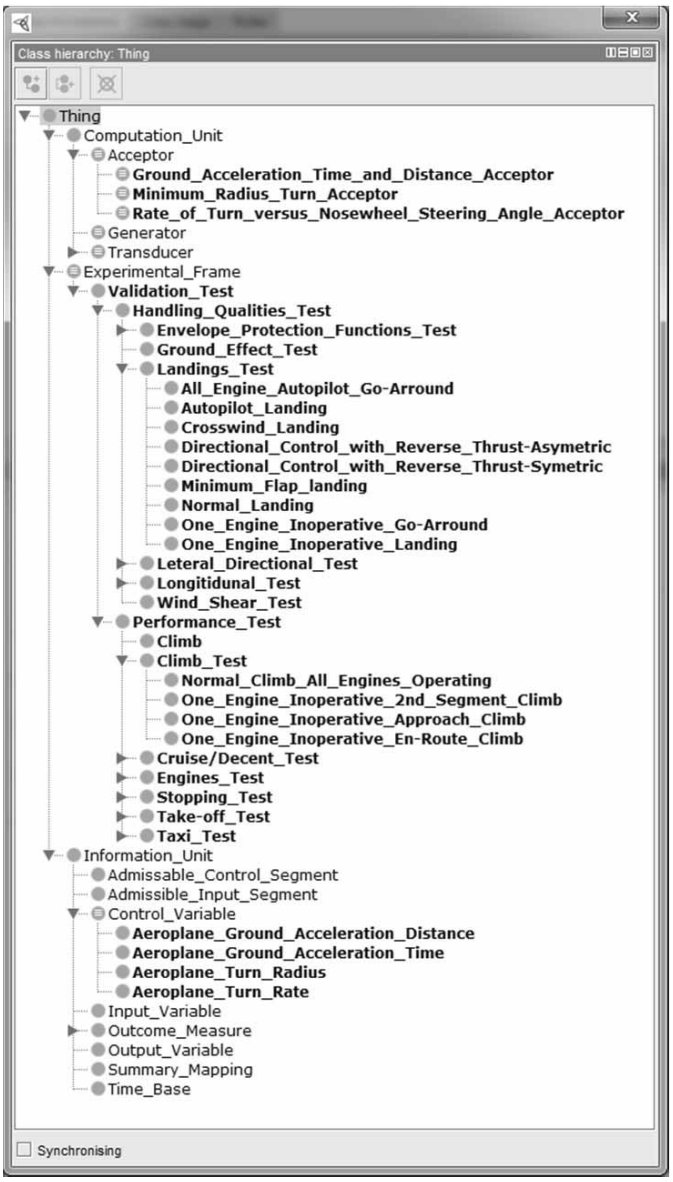


Figure 5: Objective Fidelity Evaluation Ontology Hierarchy.

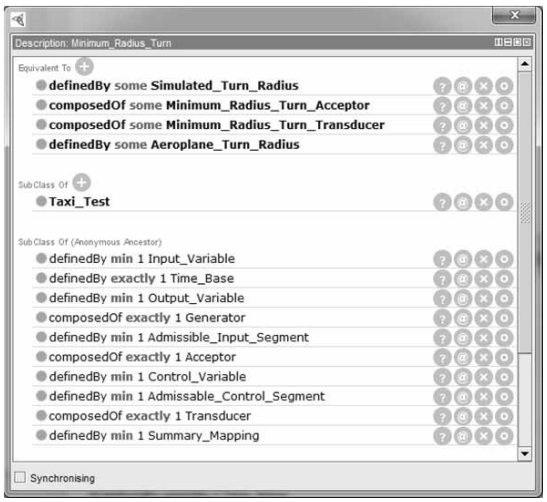


Figure 6: Minimum Radius Turn Test Description.

Figure 6 presents an example test description in Protégé. The Minimum Turn Radius Test is specified with a specific Acceptor, Transducer and Control Variables, namely Simulated Turn Radius and Aeroplane Turn Radius. On the other hand, it inherits the properties of an experimental frame. So it will also have a Generator, Input Variables, Output Variables, Admissible Input Segments, Admissible Control Segments and a Summary Mapping. It is clear that input and output variables of the flight simulator are application specific but does not vary with test cases, so generic definitions are kept for these variables and admissible segments.

Minimum Radius Turn Transducer (Figure 7) is defined with an output Simulated Turn Radius while it also inherits the properties of a Transducer. It will be using Output Variables for computing the outcome measure. Since the computation of the outcome measure is largely implementation specific, ontology does not have any knowledge about it.

As an example, the Minimum Radius Turn Acceptor is depicted in Figure 8. Since each of the tests has distinct criteria, the Acceptors will have particular inputs. Accordingly, Minimum Radius Turn Acceptor is described with Simulated Turn Radius and Aeroplane Turn Radius inputs.

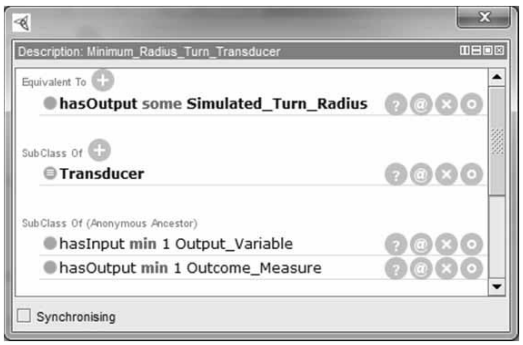


Figure 7: Minimum Radius Turn Transducer Description.

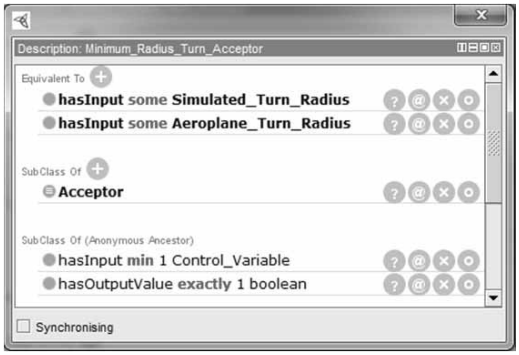


Figure 8: Minimum Radius Turn Acceptor Description.

On the other hand the output of the Acceptor is always a Boolean. It reports if the criterion is matched or not.

Semantic Web Rule Language (SWRL) [40] is used to formalize the acceptance criteria. SWRL can be regarded as an extension to OWL to specify rules for enhancing expressivity.

Thus rule-based reasoning over the knowledge captured in an ontology is possible. In this study, rules specify how the inputs of the Acceptor are used to compute if the test is successful or not. In

Figure **9**, the rule in the front windows says that Minimum Radius Turn Acceptor has a true output when the difference between the simulated and the real minimum turn radius is smaller than 20 %.

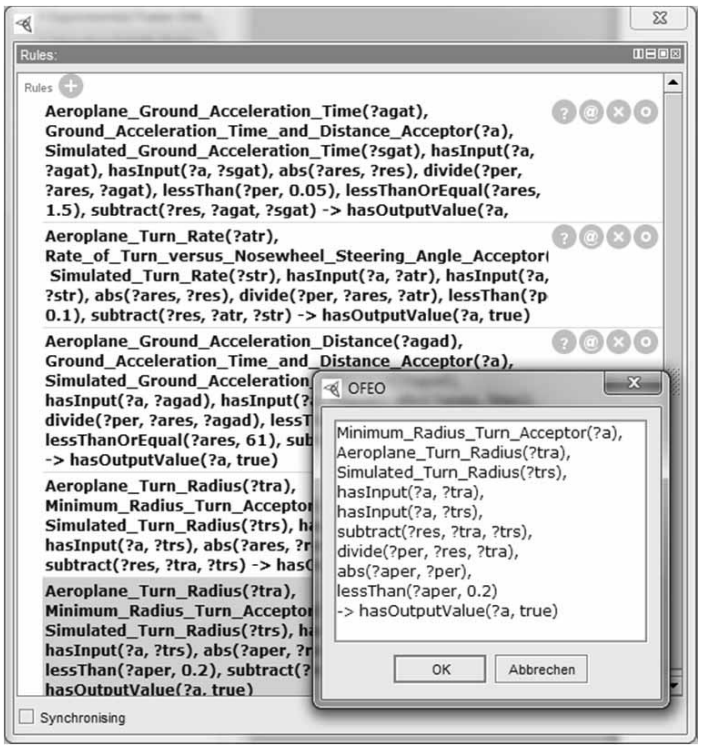


Figure 9: Rules for Acceptors.

4 Conclusion

Research simulators require flexible and adoptable test methodologies to accommodate frequent changes to their components. This paper presents an ontology based metamodeling approach for adopting a Model Based Testing methodology for objective flight simulator evaluation.

Experimental Frames Ontology adopts the concept of Experimental Frames from Discrete Event Systems Specification, as an upper ontology to specify a formal structure for test cases.

Thus with Experimental Frames, concepts of Model Based Testing could be formally specified. This established a solid base for modeling specific test cases. Then in Objective Fidelity Evaluation Ontology that builds upon Experimental Frames Ontology, domain specific meta-test definitions are modeled.

While Web Ontology Language is used as the ontology language; Semantic Web Rule Language is employed to capture the rules. Protégé is utilized as the ontology development environment.

This effort assembled the semantic infrastructure for developing model based automated test methodology for simulator fidelity evaluation. The next step is to construct the toolset for developing the test models utilizing the presented metamodels. This toolset set shall also support model transformations to generate executable test cases and execution of these test cases.

Although Web Ontology Language, Semantic Web Rule Language are employed in this metamodeling step, the representation form of the knowledge captured in ontologies may vary in toolset implementation due to practical reasons like platform compatibility.

References

[1] D. Allerton. *Principles of Flight Simulation.* John Wiley & Sons, Ltd, West Sussex, United Kingdom, 2009.

[2] P. Saager. *Real-Time Hardware-in-the-Loop Simulation for 'ATTAS' and 'ATTHeS' Advanced Technology Flight Test Vehicles*. in AGARD Guidance and Control Panel, 50th Symposium, Izmir, Turkey, 1990.

[3] S. Klaes. *ATTAS Ground Based System Simulator - An Update-*. In: AIAA Modeling and Simulation Technologies Conference and Exhibit, Denver, CO, 2000.

[4] B. Sullivan and P. Soukup. *The NASA 747-400 Flight Simulator: A Natonal Reseource fir Aviation Safety Research*. In: AIAA Flight Simulation Technologies Conference, San Diego, CA, 1996.

[5] R. Smith. *A Description of the Cockpit Motion Facility and the Research Flight Deck Simulator*. In: AIAA Modeling and Simulation Technologies Conference and Exhibit, Denver, CO, 2000.

[6] H. Duda, T. Gerlach, S. Advani and M. Potter. *Design of the DLR AVES Research Flight Simulator*. In: AIAA Modeling and Simulation Technologies (MS) Conference, Boston, MA, 2013.

[7] M. White and G. Padfield. *The Use of Flight Simulation for Research and Teaching in Acedemia*. In: AIAA Atmospheric Flight Mechanics Conference and Exhibit, Keystone, CO, 2006.

[8] S. Advani, D. Giovannetti and M. Blum. *Design of a Hexapod Motion Cueing System for NASA Ames Vertical Motion Simulator*. In: AIAA Modeling and Simulation Technologies Conference and Exhibit, Monterey, California, 2002.

[9] O. Stroosma, R. van Paassen and M. Mulder. *Using the Simona Research Simulator for Human-Machine Interaction Research*. In: AIAA Modeling and Simulation Technologies Conference and Exhibit, Austin, Texas, 2003.

[10] T. Longride, J. Bürki-Cohen, T. Go and A. Kendra. *Simulator Fidelity Considerations for Training and Evaluation of Today's Airline Pilots*. In: Proceedings of the 11th International Symposium on Aviation Psychology, Columbus, OH, 2001.

[11] D. Braun and R. Galloway, *Universal Automated Flight Simulator Fidelity Test System*. In: AIAA Modeling and Simulation Technologies Conference and Exhibit, Rhode Island, 2004.

[12] C. Wang, J. He, G. Li and J. Han. *An Automated Test System for Flight Simulator Fidelity Evaluation*. Journal of Computers, vol. 4(11), 2009.

[13] C. Wang, J. Han, G. Li and H. Jiang. *Flight Simulator Fidelity Evaluation Automated Test System Analysis*. In: 2008 International Workshop on Education Technology and Training, Shanghai, China, 2008.

[14] P. Jarvis, D. Spira and B. Lalonde. *Flight Simulator Modeling and Validation Approaches and Pilot-in-the-loop Fidelity*. In: AIAA Modeling and Simulation Technologies Conference and Exhibit, Honolulu, Hawaii, 2008.

[15] J. Zander, I. Schieferdecker and P. Mosterman, A *Taxononomy of Model-Based Testing for Embedded Systems from Multipke Industry Domains.* In: Model-Based Testing for Embedded Systems, Boca Rato, CRC Press, 2012, pp. 3-23.

[16] A. Guduvan, H. Waselynck, V. Wiels, G. Durrieu, Y. Fusero and M. Schieber. *A Meta-Model for Tests of Avionics Embedded Systems*. In: Modelsward, Barcelona, Spain, 2013.

[17] D. Gasevic, D. Djuric and V. Devedzic. *Model Driven Architecture and Ontology Development*. Springer-Verlag, Berlin, 2006.

[18] T. Gruber. *Toward Principles for the Design of Ontologies Used for Knowledge Sharing*. Int. Journal of Human-Computer Studies, vol. 43, pp. 907-928, 1995.

[19] S. Weissleder. *Test Models and Coverage Criteria forAutomatic Model-Based Test Generation with UML State Machines*. Humboldt-Universität zu Berlin, Berlin, 2010.

[20] T. Moser, G. Düee and S. Biffl, *Ontology-Based Test Case Generation For Simulating Complex Production Automation Systems*. In: SEKE 2010, San Fransisco Bay, USA, 2010.

[21] C. Nguyen, A. Perini and P. Tonella. *Ontology-based Test Generation for Multiagent Systems.* In: 7th International Joint Conference on Autonomous Agents and Multiagent Systems, Estoril, Portugal, 2008.

[22] B. Zeigler. *Multifacetted Modelling and Discrete Event Simulation.* Academic Press Professional, Inc., 1984.

[23] B. Zeigler, H. Praehofer and T. Kim. *Theory of Modeling and Simulation: Integrating discrete event and continuous complex dynamic systems*. Academic Press, Inc., 2000.

[24] M. Traoré and A. Muzy. *Capturing the dual relationship between simulation models and their context.* Simulation Modelling Practice and Theory, 14, pp. 126-142, 2006.

[25] D. Foures, V. Albert and A. Nketsa. *Simulation Validation Using the Compatibility between Simulation Model and Experimental Frame.* In: 45th Summer Simulation Multi-conference, Toronto, Canada, 2013.

[26] K. Holger, M. Horridge, M. Musen, A. Rector, R. Stevans, N. Drummond, P. Lord, N. Noy, J. Seidenberg and H. Wangl. *The Protege OWL Experience.* In: OWLED, Galway, Ireland, 2005.

[27] P. Perfect, E. Timson, M. White, R. Erdos, A. Gubbels and A. Berryman. *A Rating Scale for Subjective Assesment of Simulator Fidelity*. In: 37th European Rotorcraft Forum, Gallarate, Italy, 2011.

[28] ICAO, *Manual Criteria for the Qualification of Flight Training Devices*. ICAO, Quebec, Canada, 2009.

[29] RAeS, *Aeroplane Flight Simulation Training Device Evaluation Handbook Vol.1 Objective Testing*. RAeS, London, 2009.

[30] B. Nader and J. B. Filippi, *An Experimental Frame for the Simulation of Forest Fire Spread*. In: Proceeding of the 2011 Winter Simulation Conference, Phoenix, Arizona, USA, 2011.

[31] A. Zengin and M. Ozturk, *Formal verification and validation with DEVS-Suite: OSPF Case study.* Simulation Modelling Practice, vol. 29, pp. 193-206, 2012.

[32] O. Corcho and A. Perez, *Evaluating Knowledge Representation and Reasoning Capabilities of Ontology Specification Languages*. In: ECAI'00 Workshop on Applications of Ontologies and Problem Solving Methods, Berlin, Germany, 2000.

[33] W. Borst, J. Akkermans, A. Pos and J. Top. *The PhysSys Ontology for Physical System.* In: QR'95 Ninth International Workshop on Qualitative Reasoning, Amsterdam, Netherlands, 1995.

[34] W. Borst and J. Akkermans. *Engineering Ontologies.* International Journal of Human-Computer Studies, vol. 46 (2/3), pp. 365-406, 1997.

[35] P. Fishwick and J. Miller. *Ontologies for Modeling and Simulation: Issues and Approaches.* In: Winter Simulation Conference, Washington, DC, 2004.

[36] U. Durak, H. Oguztuzun and K. Ider. *Ontology Based Trajectory Simulation Framework.* Journal of Computing and Information Science in Engineering, vol. 8(1), March 2008.

[37] U. Durak, H. Oguztuzun and K. Ider. *Ontology Based Domain Engineering for Trajectory Simulation Reuse*. Journal of Software Engineering and Knowledge Engineering, vol. 19(8), December 2009.

[38] B. Zeigler. *Modeling & Simulation-Based Data Engineering: Introducing pragmatics into ontologies for net-centric information exchange*. Academic Press, Inc., 2007.

[39] B. Zeigler and H. Sarjoughian. *Guide to Modeling and Simulation of Systems of Systems*. Springer, 2013.

[40] I. Harrocks, P. Patel-Schneider, H. Boley, S. Tabet, B. Grosof and M. Dean. *SWRL: A Semantic Web Rule Language Combining OWL and RuleML*. W3C, Canada, 2004.

B.8 Model-Based Testing for Objective Fidelity Evaluation of Engineering and Research Flight Simulators

Umut Durak, Artur Schmidt and Thorsten Pawletta. 2015. Model-based testing for objective fidelity evaluation of engineering and research flight simulators. In *AIAA Modeling and Simulation Technologies Conference*, Dallas, TX.

AIAA 2015-2948

AIAA Aviation
22-26 June 2015, Dallas, TX
AIAA Modeling and Simulation Technologies Conference

Model-Based Testing for Objective Fidelity Evaluation of Engineering and Research Flight Simulators

Umut Durak[1]
German Aerospace Center (DLR), Institute of Flight Systems, Braunschweig, 38108, Germany

and

Artur Schmidt and Thorsten Pawletta[2] [3]
University of Wismar, Wismar, 23952, Germany

Simulator fidelity has been defined as the conformance of a flight simulator to the characteristics of the real aircraft. Objective fidelity evaluation is an engineering approach that attacks the fidelity problem with comparison of simulator and the actual system behavior over some quantitative measures. Testing can be pronounced as the fundamental mean for this comparison. From the utilization perspective, flight simulators are classified as research, engineering and training simulators. Research simulators are both test beds for flight simulator research and computational tools for flight systems and human factors research. Engineering simulators are used for systems development and training simulators are utilized for flight training. While training simulators are subject to rare or few upgrades or modifications in their lifespan, engineering simulators are under occasional and research simulators are under frequent change. The test cases to evaluate the fidelity of training simulators are guided by standards whereas for engineering and research simulators, test cases may present a great variation depending on the scope of change and the use case. These two characteristics of engineering and research simulators, combined with the complexity of today's aircrafts necessitate new methodologies for efficient and effective testing. Model-Based Testing (MBT) targets flexibility and adaptability via utilization of models for specification of test cases and proposes workflows for automatic test case generation. The paper presents an MBT approach for objective fidelity evaluation of engineering and research simulators. The proposed approach is exercised with an infrastructure implementation and an example case study. Thus, evidences are collected that indicate increased efficiency and an effective test process.

Nomenclature

A	=	Acceptor
AL	=	Autopilot Landing
BM	=	Basic Model
CP	=	Cruise Performance
DEVS	=	Discrete Event System Specification
EF	=	Experimental Frame
EM	=	Executable Model
G	=	Generator
GA	=	Ground Acceleration
ICAO	=	International Civil Aviation Organization

[1] Research Scientist, Flight Dynamics and Simulation, Lilienthalplatz 7, 38108 Braunschweig, Germany, umut.durak@dlr.de, AIAA Member.
[2] Ph.D.Student, Computational Engineering & Automation Research Group, PF 1210, D-23952 Wismar, Germany, artur.schmidt@hs-wismar.de.
[3] Professor, Computational Engineering & Automation Research Group, PF 1210, D-23952 Wismar, Germany, thorsten.pawletta@hs-wismar.de.

MB = Model Base
MBT = Model Base Testing
MUT = Model Under Test
PES = Pruned Entity Structure
SES = System Entity Structure
STF = Source Test File
SUT = System Under Test
T = Transducer
TE = Test Environment
TS = Test Scenario

I. Introduction

FLIGHT simulators have been used by aeronautics research community now for more than 30 years in developing and experimenting advanced concepts and conducting aviation human factors research. Some of the well-known early examples are ATTAS Ground Based Simulator from German Aerospace Center (DLR),[1,2] NASA Crew Vehicle Systems Research Facility in Ames Research Center[3] and Visual Motion Simulation and Cockpit Motion Facility from Langley Research Center[4]. On the other hand, Air Vehicle Simulator (AVES) of DLR,[5] HELIFLIGHT from the University of Liverpool,[6] NASA Ames Vertical Motion Simulator[7] and SIMONA of Delft University of Technology[8] can be pronounced as the well-known ones which are currently in operation.

Fidelity in flight simulation can be defined as the degree to which a flight simulator matches the characteristics of the real aircraft.[9] As well as on the training efficiency and transfer of training,[10] fidelity of the simulator has an important effect on the quality of the results of simulation experiments for research and development. Objective simulator fidelity assessment provides an engineering standard to qualify the degree of fidelity through objective measures. It approaches the fidelity problem with comparison of simulator and the actual flight over some quantitative cues. Objective simulator fidelity assessment is a tedious and labor intensive effort. Along with that, due to their intended use, engineering and research simulators are subject to a constant change. Furthermore, while the test cases to evaluate the fidelity of training simulators are guided by standards, for engineering and research simulators, test cases may present a great variation depending on the scope of change and the use case.

Regarding the inevitable changes during the lifecycle and the variability in the test cases for engineering and research simulators, combined with the complexity of today's aircrafts, it is required to develop new methodologies for efficient and effective testing. Model Based Testing (MBT) was introduced as a proposal for automating test case generation from a test specification, also called test model, instead of implementing test cases manually.[11] MBT not only automates the testing process, but also enhances the flexibility and adaptability of the testing infrastructure via automating the test case design.[12] Despite the fact that it is widely used in the software testing community, its application in modeling and simulation is quite limited.[13]

Model-based methodologies ask for metamodels to express models. Metamodeling on the other hand requires a complete and accurate specification of concepts. In this study, we referred to simulation theory to fulfill these preconditions. Experimental Frame (EF) is employed for formally specifying simulation test cases, and System Entity Structure (SES)[14] is used for metamodeling. The concept of EF originates from the Discrete Event System Specification (DEVS)[15]. The objective has been an explicit separation between a dynamic model and any experiment with it. EF formally specifies a limited set of circumstances under which a model has to be observed. SES can be defined as an ontology with a limited set of elements that are used to describe various system structures.[16]

Test cases are specified following the formal structure of EF. For generating an executable EF, configurable Basic Models (BMs) for objective fidelity evaluation are provided by a Model Base (MB). BMs usually correspond to atomic or coupled models which are used to compose modular, hierarchical models.[16] The SES is represented by a directed and labeled tree with links to BMs in the MB.

This paper is based on two previous studies: Following the methodology introduced by Schmidt et al.[14], the SES ontology is used for specification of all abstract test cases. Based on the SES and MB, a specific executable test case or a test suite is automatically generated for a flight simulation model under test. The specification of objective fidelity evaluation test cases in SES ontology are mostly adopted from Objective Fidelity Evaluation Ontology of Durak et al.[17]Utilizing the testing infrastructure from Schmidt et al.[14], implementation is carried out in MATLAB/Simulink; BMs are developed as Simulink block and SES is described using SES Toolbox for MATLAB/Simulink.[18] The approach is exemplified with the construction of a sample test suite.

The paper starts with introducing objective fidelity evaluation and automated testing. Then the model-based testing approach for simulations will be presented. The fourth section will proceed with proposing a methodology for model based testing for objective fidelity evaluation. Before the concluding remarks, a sample case study is presented.

II. Objective Fidelity Evaluation and Automated Testing

There is no consensus among researchers on a single index of measurement for simulator fidelity. Further it has strongly been related to the task to be performed with the simulator. There are two approaches to evaluate the fidelity of a simulator. These are subjective fidelity evaluation and objective fidelity evaluation. In subjective approach, user feedback is structured using rating scales to identify the degree of realism felt by the user.[19] But the individual opinions and bias of raters makes it hard to generalize the evaluations across the scales.[20] Objective approaches attack the fidelity problem with the comparison of simulator and the actual flight over some quantitative cues.

As the well accepted global standard for qualification of flight training devices, International Civil Aviation Organization (ICAO) 9625 Manual of Criteria for the Qualification of Flight Training Devices, 3rd Edition[21] specifies the set of test cases for objective validation of simulators. These test cases include comparison of the results from tests conducted in the simulator and aircraft validation data. As an example, in Ground Acceleration Time and Distance (1.b.1[21]) test, it is required to demonstrate that the time and distance required for the simulator to perform a takeoff run conform to the real aircraft. It is recommended to perform a normal takeoff ground roll and to record the time and distance from break release to rotation speed. The test mandates a conformance in time either ±1.5 sec. or ±5% and in distance ±61 m (200Ft) or again ±5%.

Additionally, Aeroplane Simulation Training Device Evaluation Handbook Vol. 1 Objective Testing[22] of the Royal Aeronautical Society (RAeS) explains the implementation of each test and introduces some example cases with some plots, thus enhances the understanding of objective tests introduced in ICAO 9625.

Automated testing can defined as the use of software to control the execution of tests and a comparison of actual outcomes to the predicted ones. Automated testing in objective fidelity evaluation is promoted by the RAeS regarding its benefits; repeatability, ease and rapidity of conducting tests. The features of an automatic testing system is introduced in the RAeS handbook as initializing the simulator with the test initial conditions, trimming the aircraft, creating the stimulus if required, using flight controls and finally checking the simulator output against test criteria. [22]

Braun and Galloway[23] reported their automated fidelity test system that compares directly the flight test results and manual execution of flight tests in simulators. Wang et al.[20, 24] presented Automated Test System that measure force function, evaluation function and transport delay with its non-intrusive interface with operator station. Both efforts on automated testing for objective flight simulator evaluation utilized fixed test descriptions and targets at automation of test case execution. With the MBT approach, we not only target at test execution automation, but also enable automated test case generation to tackle high flexibility and efficiency requirements of objective fidelity assessment for engineering and research flight simulators.

III. Model Based Testing of Simulations

A. Model Based Testing

MBT often targets the functional testing of a System Under Test (SUT) .[25] One interpretation of MBT is shown in Fig.1. Model driven approaches suggest deriving a formal systems model based on the system requirements. System model represents a simplification of the structural and behavioral relationships of the components. In the next step, executable components can be generated from the formal system model. These components form the SUT. Model based testing promotes that the same system requirements are used to derive a formal test model. They describe the intended behavior of the SUT that needs to be tested. From the test model, a specific test case or a collection of test cases, called a test suite, can be generated for an SUT.[26] The idea is that test cases are abstracted in a test model, and then an MBT tool is employed to generate a set of test cases from that model.[12]

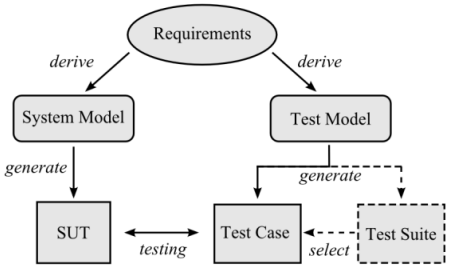


Figure 1 Model Based Testing approach[26]

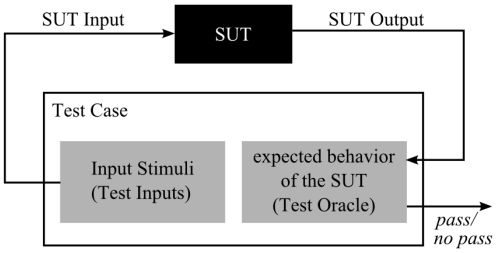


Figure 2 Structure of a Test Case

Weissleder defines a test case as the combination of a sequence of input stimuli to be fed into a SUT, called test inputs, and the expected behavior of a SUT (Fig.2). [25] The expected behavior is often produced using a test oracle. A test oracle contains a judgment unit to make a pass or no pass decision.

MATLAB/Simulink is a popular environment for model based systems development from MathWorks, Inc.[27] It provides a graphical editor, customizable block libraries and numerical solvers for modeling and simulation of dynamic systems. It is widely utilized in flight simulation model development for AVES. Hereby, in this study, we adopt the MBT practices from software testing and propose a testing methodology for flight simulation models developed in MATLAB/Simulink.

Between 2005 and 2008 Zander developed early ideas of employing MBT in a MATLAB/Simulink environment.[28,29] Her Model-in-the-Loop for Embedded System Test – Test Harness (MiLEST) infrastructure provides well-structured libraries for test data generation, test control and test validation functions. MathWorks on the other hand has been providing Simulink Verification and Validation[30] since 2006 for the realization of MBT in MATLAB/Simulink. As in the MiLEST, Simulink Verification and Validation is also providing library blocks that target test functions. Both of this efforts aim at providing a methodology to test the controller models that will be used to generate code to be deployed in an embedded target. Thus, the model that is subject to a test in these two approaches is not necessarily a simulation model or a flight simulation model. In this paper, we propose an MBT approach based on the system theoretical methodologies that are adopted in the simulation theory. Before the concluding remarks, a prototype infrastructure implementation and sample case study will be adduced to make the evidences of applicability traceable.

B. Experimental Frame and System Entity Structure and Model Base Framework

The concept of the Experimental Frame and the System Entity Structure and Model Base (SES/MB) framework was introduced by Zeigler and his colleagues as a part of their system theoretical based approaches for modeling and simulation, DEVS specification.[15,16]

An EF specifies a limited set of circumstances under which a model has to be observed. Following Zeigler,[15] the formal specification of EF is given by the 7-tupel:

$$\mathbf{EF = < T, I, O, C, \Omega i, \Omega c, SU >}$$

where:
T is the time base,
I is the set of input variables,
O is the set of output variables,
C is the set of control variables,
Ωi is the set of admissible input segments,
Ωc is the set of admissible control segments and
SU is a set of summary mappings.

Reader can refer to Traoré et al.[31] for further definition of EFs.

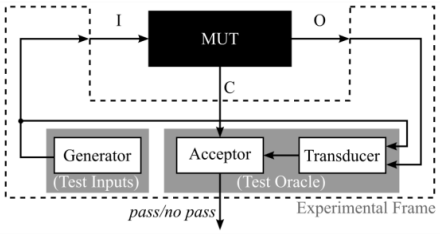


Figure 3 Realization of Experimental Frame

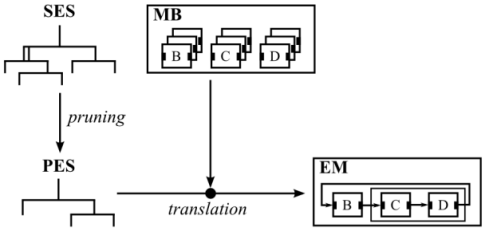


Figure 4 SES/MB Framework[16]

While the EF can be implemented in various ways, Zeigler[15] recommends the implementation of EF as a coupled model, consisting of a generator, acceptor and a transducer, which are connected to the model. We call this model, Model Under Test (MUT). The realization of EF coupled to an MUT is presented in Fig.3 schematically. Test inputs are produced by the generator. The acceptor and transducer form a test oracle. Based on the output variables, the transducer calculates outcome measures in the form of performance indices, comparative values, statistics etc. The acceptor corresponds to a decision unit that decides if an experiment is valid or not. It monitors its inputs and maps them to a specified admissible control segment. The experiment is invalid in the case of a violation of the admissible control segment.

The SES is represented by a directed and labeled tree with links to BMs in the MB. MB can be defined as a repository for BMs that describe the dynamic behavior and represent atomic or coupled systems. Moreover, a set of elements and axioms have been provided in SES to describe system structures.[32] These elements include four node types: (i) entity, (ii) aspect, (iii) specialization and (iv) multiple aspect. Entity represents real or artificial system components. The other node types describe the relationships between their parent and child entities. While aspect nodes denote the decomposition relationship of an entity, specialization nodes represent the taxonomy of an entity. The multiple aspect nodes, finally, represent a multiplicity relationship which specifies that the parent entity is a composition of multiple entities of the same type. Furthermore, specific suffixes are used for a clear separation of the node types. All aspect nodes have the suffix *Dec*, specialization nodes the suffix *Spec* and the multiple aspect have the suffix *MAsp*. Nodes without the defined suffixes correspond to entity nodes.

Figure 4 shows the fundamental structure of the SES/MB framework. The framework combines the SES ontology with the classical workflow of modeling and the simulation of modular, hierarchical systems.[16] It promotes the methodologies for an automatic generation of an executable simulation models using the specification of the system structure in SES and the executable model components in MB.

Pruning is the operation by which a distinct system structure can be derived from an SES. The result is called Pruned Entity Structure (PES). SES Variables represent a kind of user interface and are the basis for the pruning operation. There are two types of SES Variables: (i) related to the system structure and (ii) related to the parameter setting of the nodes. After pruning, *translation* operation is conducted to generate an executable simulation model (EM) based on the information of the PES and BMs from the MB.

C. Proposed Approach

We propose to employ SES/MB for MBT of simulation models. An SES needs to be constructed for specifying the test case structure based on EFs. The proposed top level SES based upon our previous study[17] is as presented in Fig. 5. The node TestScenarioDec indicates the decomposition of entity TestScenario in the two entities (i) MUT and (ii) EF. Referring to Fig.3, an EF consists of a generator (G), an acceptor (A) and a transducer (T). In SES, attributes can be attached to any node. Aspect nodes, such as TestScenarioDec and EFDec, define the coupling relationship between their direct predecessors and successors as attributes. The tuple (MUT.out, EF.in) shows that the output of the MUT is connected with the input of the EF, etc. The entity nodes G, A and T needs to be specialized by their successor nodes, GSpec, ASpec and TSpec and application specific generators, acceptors and transducers need to be define as leaf nodes. Then in these leaf nodes, it is required to define attributes that references to BMs in the MB and the parameter setting of BMs.

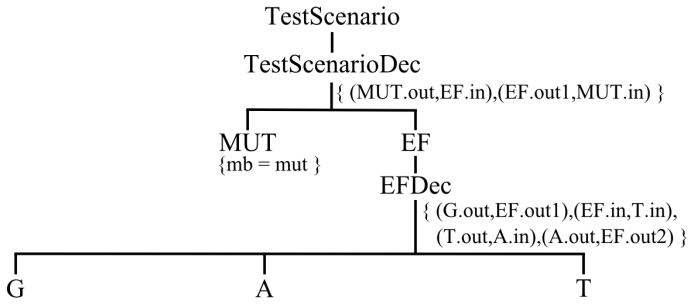


Figure 5 SES structure for EF

The MB will provide a set of BMs, which map different types of generators (G), acceptors (A) and transducers (T). MB will also contain MUTs for the generation of executable test scenarios.

As presented in Fig. 6, Source Test Files (STF) are introduced as scripting interface for the Test Environment (TE). Using STF, the user can specify the MUTs and the test cases that shall be performed on the MUTs. A specific MUTi and EFj will be selected via value assignments to the SES Variables in the STF. TE then will interpret the STF for each test case. First, for each MUTi, the corresponding set of EFs will be identified. Then for each EFj from this set, a specific SES Variable configuration will be picked and sent to the SES/MB framework. The SES/MB framework will generate an executable test scenario (TS) as a coupled system of MUTi and the EFj. The TE then will execute the TS; collect the actual test results and proceeds with the next EFj. After all the specified EFs are executed on the MUTi, the next MUTi and EF set will be selected from the STF and the test cycle will run again. Finally, the results of all the tests will be interpreted by the Test Evaluator. This component is proposed to compute additional statistics, prepare documentation and present the results to the user.

The prototype infrastructure implementation of the proposed approach is done in MATLAB/Simulink and validated using a test case from robotics domain in our previous study.[14] The following section will try to present how this proposed approach can be employed for objective validation of flight simulation models.

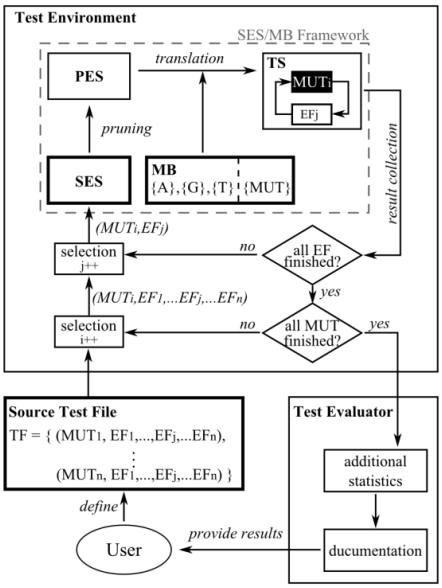


Figure 6 Testing Infrastructure based on SES/MB Framework[14]

IV. Model Based Testing for Objective Fidelity Evaluation

To apply the proposed approach for objective validation, based on our ontology for objective flight simulator fidelity evaluation,[17] an SES has been constructed. A simplified excerpt from this SES has been depicted in Fig. 7. In this graph, *G* is specialized in an entity *Ground Acceleration (GA)*, *Autopilot Landing (AL)* or a *Cruise Performance* (*CP*), *T* in *Ground Acceleration Time* (*GA_Time*) or *Ground Acceleration Distance* (*GA_Distance*) and *A* in *Initial* or *Continuation*.

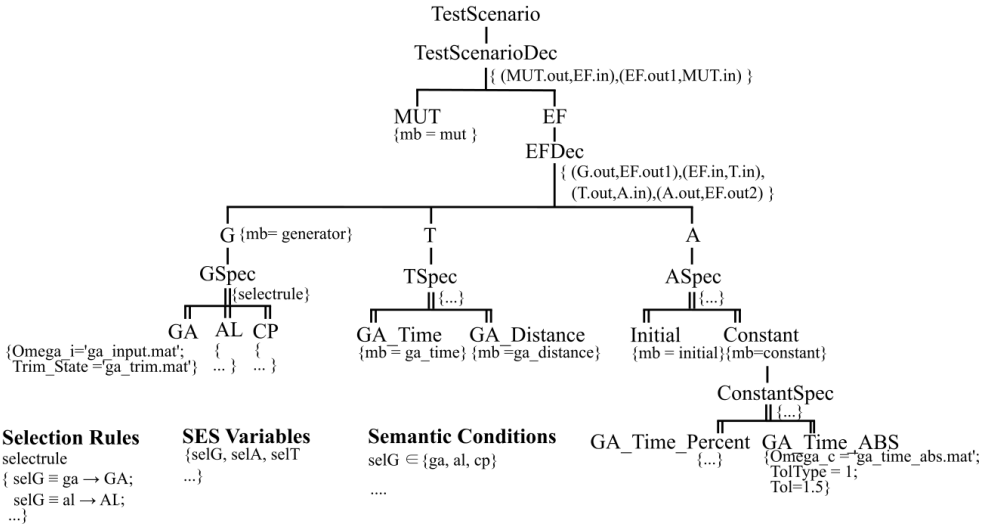


Figure 7 A Simplified Excerpt from Objective Validation SES

A test case that is specified in ICAO 9625 is capture in SES with a generator, transducer and an acceptor. Generators references to a BM in the MB that basically initializes the model with the defined trim file and applies the defined inputs to the model. While the reference to the MB is specified as the attribute of G, the inputs and trim files is cited in the leaf nodes. For GA, the inputs are defined with the attribute *Omega_i* which references to *ga_input.mat* file and trim file is defined with attribute *Trim_State* that references to *ga_trim.mat* file. Transducers interpret the outputs of the model and compute the outcome measures that will be subject to comparison for validity. Since every test case defined in ICAO 9625 possesses its own outcome measure, in SES a transducer is defined for each test case. *GA_Time* refers to the transducer for Ground Acceleration test and computes time to reach rotate speed, Vr, from break release. The BM that conducts this computation, namely *ga_time*, is referred in *mb* attribute of *GA_Time* entity. Acceptors decide upon the validity of the experiment by comparing the outcome measures with the admissible control segments. In case of flight simulation, admissible control segments are flight test data and tolerances to be checked against have been defined in the ICAO 9625. There may be various kinds of acceptors but two of the well applied ones can be seen in Fig. 7. Initial refers to BM in the MB that compares the initial value and Constant refers to the BM that makes the comparison for a constant value. In the specializations of these entities, the admissible control segments and tolerance are defined as attributes. As an example the acceptor entity, Ground Acceleration Time Absolute Value (*GA_Time_ABS*) indicates that the control segment is given in *ga_time_abs.mat* file in it attribute *Omega_c*. Referring to the previous explanation of Ground Acceleration Time and Distance test, it specifies the tolerance type as absolute value with setting *TolType* attribute to 1 and sets the tolerance as 1.5 by *Tol* attribute, which means ±1.5 sec conformance in time between the simulator and the real aircraft.

$$\textbf{SES\_Vars = \{selG=ga, selT=ga\_time, selA=ga\_time\_abs\}} \quad (1)$$

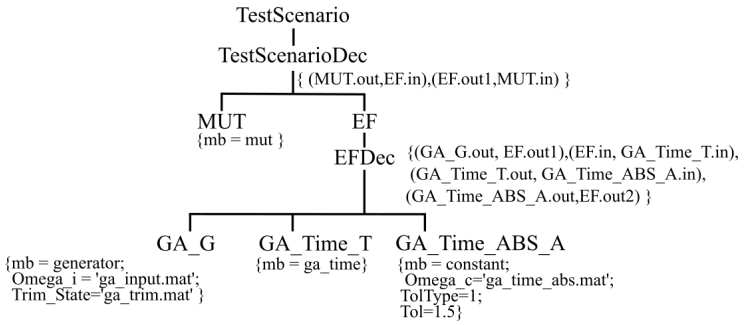


Figure 8 PES Example

As already introduced, SES Variables are utilized as a kind of user interface for the pruning operation. The *pruning* operation targets at deriving a decision-free tree, called PES, with corresponding parameter settings from an SES. Before the pruning operation, all SES Variables must be assigned a value from the range set specified in Semantic Conditions. By evaluating the SES Variables and Selection Rules, all variability in structure and parameter setting will be resolved during the pruning operation. As an example following the SES Variable configuration given in (1) will lead to PES that is depicted in Fig. 8.

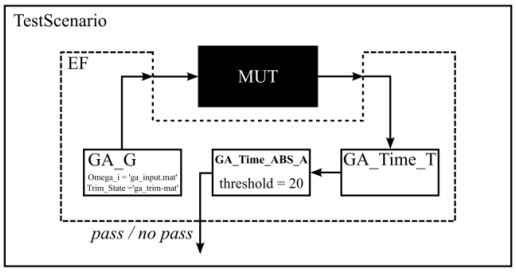


Figure 9 Generated executable test scenario based on the PES and the corresponding MB

The required information to generate an executable test scenario, such as references to the BMs in the MB, their parameter setting and the modular, hierarchical structure as well as the coupling relationships is available in the derived PES. The translation operation is carried out by scripts that accesses all the required information from the PES tree and uses the BMs in MB to generate an executable test scenario. The representative structure for the generated executable test scenario using the PES provided in Fig. 8 is presented in Fig.9.

V. The Prototype Infrastructure Implementation and an Example Case Study

The prototype infrastructure implementation aims at exercising the presented approach in order to collect evidences of its applicability. In this prototype implementation an SES has been constructed that targets a small subset of test cases that are defined in ICAO 9625. This subset includes Ground Acceleration Time and Distance (1.b.1[21]), Autopilot Landing (2.e.5[21]) and Cruise Performance (1.d.3[21]). Generators, transducers and the acceptors are specified for these tests using the MATLAB frontend. The prototype implementations of the corresponding BMs have been conducted and a representative MB has been constructed. The implementation of the automation scripts for test case generation is prototyped. And finally a sample test case generation is exercised.

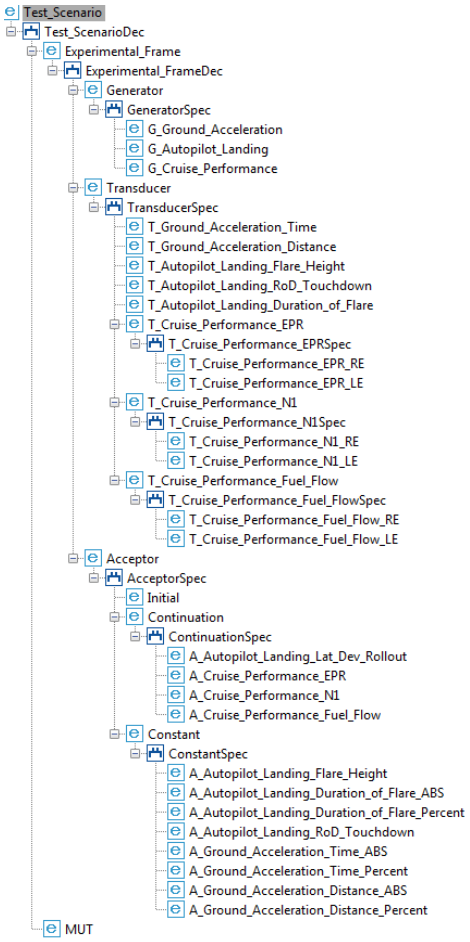


Figure 10 Sample SES Implementation

	Source Comp.	From Port	Sink Comp.	To Port
1	Experimental_Frame	1	Transducer	1
2	Generator	1	Experimental_Frame	1
3	Generator	1	Transducer	2
4	Transducer	1	Acceptor	1

Figure 11 Sample Couplings of the Experimental_FrameDec node

	Attribute Name	Value
1	Omega_c	'cruise_performance_n1.mat'
2	TolType	0
3	Tol	0.03
4	ResultsFile	'cruise_performance_n1_results.mat'

Figure 12 Sample Attributes of the A_Cruise_Performance_N1 entity

	Selection	Condition
1	A_Autopilot_Landing_Lat_Dev_Rollout	Acc_Choice == 21
2	A_Cruise_Performance_EPR	Acc_Choice == 22
3	A_Cruise_Performance_N1	Acc_Choice == 23
4	A_Cruise_Performance_Fuel_Flow	Acc_Choice == 24

Figure 13 Sample Selection Conditions of ContinuationSpec node

	Variable Name	Variable Value
1	Acc_Choice	21
2	Tra_Choice	6
3	Gen_Choice	1
4	Acc_Type_Choice	2
5	Engine_Choice	0

Figure 14 Sample SES Variables

	Semantic Conditions
1	ismember(Gen_Choice,[1:3])
2	ismember(Tra_Choice,[1:8])
3	ismember(Acc_Type_Choice,[1:3])
4	(Acc_Type_Choice == 1) | (Acc_Type_Choice == 2 & ismember(Acc_..
5	ismember(Engine_Choice,[0,1])

Figure 15 Sample Semantic Conditions

Figure 10 presents the SES, which is created using the SES Toolbox for MATLAB/Simulink[18]. Test Scenario is decomposed into the experimental frame and the MUT, which in this case is a Flight Dynamic Model. Experimental frame is decomposed into a generator, transducer and an acceptor. A sample coupling is depicted in Fig. 11 for the elements of the experimental frame. Attributes are defined for each entity. Fig.12 exemplifies this for the acceptor for checking N1 value in the cruise performance test (*A_Cruise_Performance_N1*). It says that the reference value, control segment, will be read from a mat file name *cruise_performance_n1.mat*. The tolerance type is relative tolerance, specified by the value *0* and value of the tolerance *%3*. It also specifies a file that the results shall be recorded.

Specialization nodes enable to define different kinds of the parent entity. Specific generators, transducers and accepters are structured using specialization nodes. As an example, reader can follow from Fig. 10 that three different generators are defined for ground acceleration, autopilot landing and cruise performance tests. Selection conditions are used to set the specific configuration for a particular test case. In other words, as shown in Fig. 13, the conditions of selecting a particular child entity of a specialization node are specified. Selection values that are

presented in Fig. 14 are then used during pruning. The selection variables (SES Variables) are constraint by semantic conditions which specify the valid ranges of the variables. An extract from the semantic conditions is given in Fig. 15.

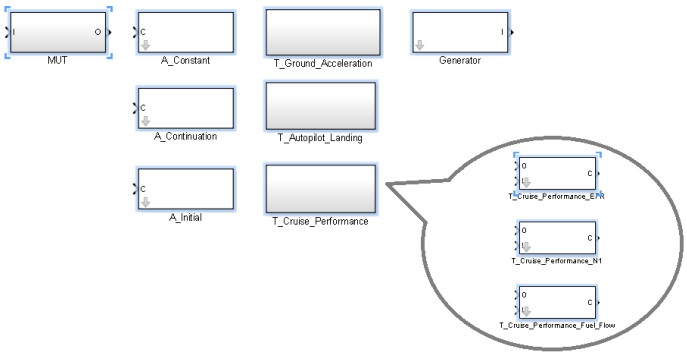


Figure 16 An Excerpt from Model Base

Figure 16 presents an excerpt from the MB which contains the blocks that the test case is made up of. Source Test File is then scripted as exemplified below in Fig. 17 for setting SES variables, pruning the SES and generating the test model via translation.

```
% Source Test File

%% Load a SES
objSES = load('..\SES_DLR\AVES_Objective_Validation.mat');
objPES = pes(objSES.new);

%% Set SES Variables
SES_Vars = { {'Acc_Choice',31},...
             {'Acc_Type_Choice',3},...
             {'Tra_Choice',6},...
             {'Engine_Choice',1},...
             {'Gen_Choice',3} };

%% Pruning
objPES.prune(SES_Vars);

%% Translation
modelTestcase = new_system(objPES.getRootPath());
open_system(modelTestcase);
modelBase = load_system('..\Data_Base_and_Model_Base_DLR\AVES_Objective_Validation_Model_Base.mdl');
modelTranslator(objPES,modelTestcase,modelBase);
```

Figure 17 Example Test Source File

With the SES variables specified in Test Source File, the test case is automatically generated. The *MUT* is copied and connected to an *Experimental_Frame* block. In this block, appropriate generators, transducer and acceptor are copied from the MB and connections are made using the coupling specifications in SES. Figure 18 presents the automatically generated test scenario in MATLAB/Simulink.

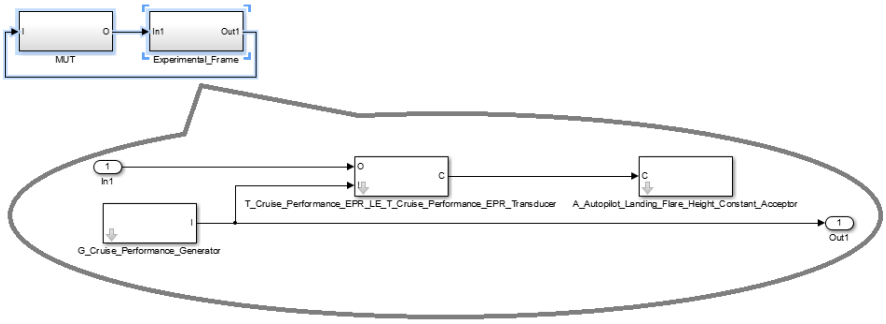


Figure 18 Automatically Generated Test Case

VI. Conclusion

The fidelity of the engineering and research simulators has an important effect on the quality of the results of simulation experiments for research and development. Furthermore their fidelity evaluation is more challenging then training simulators since engineering and research simulators are subject to more frequent change and the test cases for engineering and research simulators may present a great variation depending on the scope of change and the use case. These two characteristics of engineering and research simulators, combined with the complexity of today's aircrafts require more efficient and effective testing methodologies.

In this study, Model-Based Testing approach is presented for flight simulator objective fidelity evaluation. The approach is developed based on Experimental Frame and, System Entity Structure and Model Base Framework. Test models are specified using System Entity Structure and transformed into executable tests employing components from a Model Base that consists of basic blocks for Experimental Frames. Thus, not only the execution of the test cases, but also the generation of the test cases is automated.

As the Model Base that encompasses Experimental Frame components constitutes a reusable asset library for model testing, the adaptability is fostered. The transformation tool automatically generates executable test cases from a test specification model. Hereby, it advances the efficiency and the effectiveness.

A prototype infrastructure implementation is carried out in MATLAB/Simulink. The proposed approach and developed infrastructure is exercised in a case study. A sample SES is constructed for a small subset of objective tests described in ICAO 9625 as well as the implementation of the corresponding Basic Models that constitutes a sample Model Base. As an example, a test case is automatically generated. With the implementation and the test case, valuable evidences are collected. Whilst efficiency is implied by the success of automation in test case generation, effectiveness of the test approach is indicated by the effectual employment of the reusable asset library.

As the applicability and the productivity of the approach are attested, the next step is planned for the development of a production SES and Model Base that will composed of a wide selection of test cases from ICA9625. Then the infrastructure is planned to be released to the flight dynamics model developers in DLR Institute of Flight Systems.

References

[1]Saager, P., "Real-Time Hardware-in-the-Loop Simulation for 'ATTAS' and 'ATTHeS' Advanced Technology Flight Test Vehicles," *AGARD Guidance and Control Panel*, 50th Symposium, 22-25 May, Izmir, Turkey, 1990.

[2]Klaes, S., "ATTAS Ground Based System Simulator -An Update-," *AIAA Modeling and Simulation Technologies Conference and Exhibit*, Denver, CO, 2000.

[3]Sullivan, B. and Soukup, P., "The NASA 747-400 Flight Simulator: A National Resource for Aviation Safety Research," *AIAA Flight Simulation Technologies Conference*, San Diego, CA, 1996.

[4]Smith, R., "A Description of the Cockpit Motion Facility and the Research Flight Deck Simulator, " *AIAA Modeling and Simulation Technologies Conference and Exhibit*, Denver, CO, 2000.

[5]Duda, H., Gerlach, T., Advani, S. and Potter, M., "Design of the DLR AVES Research Flight Simulator," *AIAA Modeling and Simulation Technologies (MST) Conference*, Boston, MA, 2013.

[6]White, M. and Padfield, G., "The Use of Flight Simulation for Research and Teaching in Academia," *AIAA Atmospheric Flight Mechanics Conference and Exhibit*, Keystone, CO, 2006.

[7]Advani, S., Giovannetti, D. and Blum, M., "Design of a Hexapod Motion Cueing System for NASA Ames Vertical Motion Simulator," *AIAA Modeling and Simulation Technologies Conference and Exhibit*, Monterey, CA, 2002.

[8]Stroosma, O., van Paassen, R. and Mulder, M., "Using the Simona Research Simulator for Human-Machine Interaction Research," *AIAA Modeling and Simulation Technologies Conference and Exhibit*, Austin, TX, 2003.

[9] Rehmann, A. J., Mitman, R. D., and Reynolds, M. C., " A Handbook of Flight Simulation Fidelity Requirements for Human Factors Research," Crew System Ergonomics Information Analysis Center, Wright-Patterson Air Force Base, OH, 1995.

[10]Longride, T., Bürki-Cohen, J., Go, T. and Kendra, A., "Simulator Fidelity Considerations for Training and Evaluation of Today's Airline Pilots," *Proceedings of the 11th International Symposium on Aviation Psychology*, Columbus, OH, 2001.

[11]Zander, J., Schieferdecker, I. and Mosterman, P., "A Taxonomy of Model-Based Testing for Embedded Systems from Multiple Industry Domains," *Model-Based Testing for Embedded Systems*, Boca Rato, CRC Press, 2012, pp. 3-23.

[12]Utting, M., and Legeard, B., *Practical Model-Based Testing: A Tools Approach*, Morgan Kaufmann Publishers Inc., 2007.

[13]Hollmann, D., A., Cristia, M. and Frydman, C., "Adapting Model-Based Testing Techniques to DEVS Models Validation," *Proceedings of the 2012 Symposium of Theory of Modeling and Simulation – DEVS Integrative M&S Symposium*, San Diego, CA, 2012.

[14]Schmidt, A., Durak, U., Rasch, C., and Pawletta, T., "Model-Based Testing Approach for MATLAB/Simulink using System Entity Structure and Experimental Frames," *International Workshop on Model-driven Approaches for Simulation Engineering*, SpringSim'15, Alexandria, VA, 2015.

[15]Zeigler, B., P., *Multifaceted Modelling and Discrete Event Simulation*, Academic Press Professional Inc., London, 1984.

[16]Zeigler, B., P., Praehofer, H., Kim, T., G., *Theory of Modeling and Simulation: Integrating Discrete Event and Continuous Complex Dynamic Systems*, 2nd ed., Academic Press, 2000.

[17]Durak, U., Schmidt, A., Pawletta, T., "Ontology for Objective Flight Simulator Fidelity Evaluation," *SNE Simulation Note Europe*, ARGESIM/ASIM pub. TU Vienna, Vol. 24, No. 2, 8/2014, pp. 69-78.

[18]Pawletta, T., Pascheka, D., and Schmidt, A., "Ontology-Assisted System Modeling and Simulation within MATLAB/Simulink," *SNE Simulation Note Europe*, ARGESIM/ASIM pub. TU Vienna, Vol. 24, No. 2, 8/2014, pp. 59-68.

[19]Perfect, P., Timson, E., White, M., Erdos, R., Gubbels, A. and Berryman, A., "A Rating Scale for Subjective Assessment of Simulator Fidelity," *37th European Rotorcraft Forum*, Gallarate, Italy, 2011.

[20]Wang, C., He, J., Li, G., and Han, J., "An Automated Test System for Flight Simulator Fidelity Evaluation", *Journal of Computers*, Vol. 4, No. 11, 2009, pp. 1083-1090.

[21]International Civil Aviation Organization, "Manual Criteria for the Qualification of Flight Training Devices", 3th ed., ICAO, Quebec, Canada, 2009.

[22] Royal Aeronautical Society, "Aeroplane Flight Simulation Training Device Evaluation," *Handbook Vol.1 Objective Testing*, RAeS, London, 2009.

[23]Braun, D., and Galloway, R., "Universal Automated Flight Simulator Fidelity Test System," *AIAA Modeling and Simulation Technologies Conference and Exhibit*, Rhode Island, 2004.

[24]Wang, C., Han, J., Li, G., and Jiang, H., "Flight Simulator Fidelity Evaluation Automated Test System Analysis," *2008 International Workshop on Education Technology and Training*, Shanghai, China, 2008.

[25]Weissleder, S., "Test Models and Coverage Criteria for Automatic Model-Based Testing Generation with UML State Machines," Ph. D. Dissertation, Humboldt University zu Berlin, 2010.

[26]Roßner, T., Brandes, H., Götz, H. and Winter, M., *Basiswissen Modellbasierter Test*, dpunkt.verlag, Heidelberg, 2010.

[27]The MathWorks, Inc., "Simulink: Getting Started Guide," [online database], URL: http://www.mathworks.com/help/pdf_doc/simulink/sl_gs.pdf [cited: 13 May 2015].

[28]Zander, J., "Model-based testing of Real-Time Embedded Systems in the Automotive Domain," Ph. D. Dissertation, TU Berlin, 2008.

[29] Zander, J., "Model-in-the-Loop for Embedded System Test – Test Harness," [online database], URL: http://www.mathworks.com/matlabcentral/fileexchange/44328-model-in-the-loop-for-embedded-system-test-test-harness [cited: 13 May 2015].

[30]The MathWorks, Inc., "Model-Based Testing," [online database], URL: www.mathworks.de/discovery/model-based-testing.html, [cited: 13 May 2015].

[31] Traoré, M. and Muzy, A., "Capturing the dual relationship between simulation models and their context," *Simulation Modelling Practice and Theory*, Vol. 14, No.2, 2006, pp. 126-142.

[32]Zeigler, B., P. and Hammonds, P., E., *Modeling and Simulation-Based Data Engineering: Introducing Pragmatics into Ontologies for Net-Centric Information Exchange*, Academic Press, 200.7

B.9 Simulation Deployment Blockset for MATLAB/Simulink

Umut Durak, Anil Ozturk and Mehmet Katircioglu. 2016. Simulation deployment blockset for MATLAB/ Simulink. In *Proceedings of the Symposium on Theory of Modeling & Simulation: DEVS Integrative M&S Symposium*, Pasadena, CA, 630-637.

Simulation Deployment Blockset for MATLAB/Simulink

Umut Durak
DLR Institute of Flight Systems
Lilienthalplatz 7 38108
Braunschweig, Germany
umut.durak@dlr.de

Anil Öztürk
TAI - Turkish Aerospace Industries, Inc.
Fethiye Mh. Havacilik Blv.
No:17 06980 Kazan
Ankara, Turkey
anil.ozturk@tai.com.tr

Mehmet Katircioglu
TAI - Turkish Aerospace Industries, Inc.
Fethiye Mh. Havacilik Blv.
No:17 06980 Kazan
Ankara, Turkey
mehmet.katircioglu@tai.com.tr

ABSTRACT
Model-based approaches are being employed more and more in simulation development. Graphical modeling languages and code generation technologies are enabling agile model development workflows, so that simulation modelers can update their models more easily. However, the process from changing the model to releasing a new simulation version is overlooked. Simulation deployment can be defined as a collection of activities, including model checking, Model-in-the-Loop testing, code generation, build, Software-in-the-Loop testing, deployment, when applicable Processor-in-the-Loop and Hardware-in-the-Loop testing and release. When it is conducted manually and ad hoc, it is repetitive, labor intensive, time-consuming and error prone. The automation of deployment pipeline, on the other hand, requires extensive scripting, unfortunately, in way in which simulation modelers are usually not accustomed. Causal Block Diagrams propose a graphical modeling language that is extensively used in simulation of technical systems. MATLAB/Simulink supports them as the basic modeling language. Exploiting the competence of MATLAB/Simulink users on Causal Block Diagrams, this paper presents a model-based approach for automating the simulation deployment activities. Thus, rather than scripting, the deployment automation functions are made available and accessible to the simulation modelers within the graphical modeling environment that they are using.

Author Keywords
Simulation Deployment; Continuous Delivery; Model-Based Simulation Systems Engineering

ACM Classification Keywords
I.6.7 SIMULATION AND MODELING (Simulation Support Systems).

1. INTRODUCTION

Simulation systems engineering can be defined as a type of systems engineering that particularly applies the principles of simulation science, systems science and computer science and, in a broad sense, mathematics and engineering for developing, maintaining and employing simulations [9]. Model-based Simulation Systems Engineering (MBSSE) can be introduced as applying model-based engineering principles to the engineering of simulation systems. In the last couple of decades, MBSSE has become an industry wide standard paradigm for the simulation of technical systems. Model-based approaches are characterized by a seamless utilization of graphical models for specification, design and implementation [26]. MBSSE generalizes the practice of developing executable models in graphical modeling environments for system specification and design. Implementation is then carried out by employing code generation facilities of the modeling environment. Mathworks, Inc. refers to this approach as Model-Based Design (MBD) [30].

This practice provides agile development workflows for complex system models, so that simulation modelers can construct and update their models more easily. However, the process from changing the model to releasing a new simulation version has been overlooked. Not because it is not important, but it has been regarded mostly as peripheral to the overall problem with respect to the core challenges of model development. According to simulation engineering lifecycle studies, starting from the early efforts of Balci [3] to the most recent standardization efforts, resulting in the IEEE Recommended Practice for Distributed Simulation Engineering and Execution Process (DSEEP) [15], the deployment process is almost invisible.

Deployment can be defined as the activities from development to release [14]. In software development, it includes the build, deploy, test and release steps. In MBSSE, it consists of a collection of activities, including model checking, Model-in-the-Loop (MIL) testing, code generation, build, Software-in-the-Loop (SIL) testing, deployment, when applicable Processor-in-the-Loop (PIL) and Hardware-in-the-Loop (HIL) testing and release. It should be noted that, in many cases, simulation is deployed in multiple target platforms. When deployment is conducted manually and ad hoc, it is repetitive, labor-intensive, time-consuming and error prone. Typical antipatterns include inadequate revision control, manual code generation and integration practices, multiple development environment utilization, reliance on manual test setups, unrepeatable

SpringSim-TMS/DEVS, 2016 April 3-6, Pasadena, CA, USA

release pipeline and unpredictable release behavior in the target environment. The automation of a deployment pipeline requires extensive scripting for various tasks, such as triggering model checkers, model compilers and code compilers, configuring test setups, and conducting source control or file system operations.

While scripting for tool automation is commonly employed in software development, especially with the rise of model-based practices, simulation development largely depends upon utilizing graphical modeling approaches. Thereby, the simulation modelers are usually not used to scripting, particularly for tool automation.

Causal Block Diagrams (CBD) propose a graphical modeling language that is supported by many simulation development environments, such as MATLAB/Simulink, as the basic modeling language. They are extensively exploited in simulation of technical systems. Relying on the competence of MATLAB/Simulink users in CBD, this paper presents a MATLAB/Simulink blockset for automating simulation deployment activities. Thus, the deployment automation functions are made available and accessible to the simulation modelers within the graphical modeling environment.

In order to provide a concrete description of the complexity of a simulation deployment, the paper will first present Turkish Aerospace Industries, Inc. (TAI) Indigenous Rotorcraft Simulation (TIRS) and its targets. These targets will constitute a typical example from the industry. Then, a general overview of automated deployment is introduced. After explaining the basic concepts about CBD, the atomic blocks of simulation deployment will be revealed. Subsequently, the implementation of these blocks as a MATLAB/Simulink blockset will be presented with an example case study.

2. SIMULATION DEPLOYMENT: TIRS CASE

2.1. TAI Indigenous Rotorcraft Simulation

MBD has enabled the aeronautics community with agile model development workflows and has promoted rapid iterations over aerodynamic configuration and flight control systems design. Model development has been integrated to product development by employing mature code generation practices [23]. Constant quantitative evaluation of the current air vehicle design has provided valuable opportunities to the designers for further optimization and tuning. As such, TAI have been developing their MBD environment, known as TIRS, and utilizing it in their ongoing rotorcraft development programs.

TIRS is an in-house tool being developed by TAI to support rotorcraft design activities, including flight mechanics design and analysis, handling quality analysis, Automatic Flight Control System (AFCS) design as well as real-time flight simulation. The development of TIRS is based on the physical modeling of all the rotorcraft components individually in a modular structure. Then, the contribution of each component to the equations of motion is calculated based on the detailed rotorcraft characteristics. The principle model structure consists of various modules, including equations of motion, main rotor, tail rotor, fuselage, empennage, actuators and landing gear. All TIRS modules are developed in MATLAB script files (m-files). TIRS uses Simulink Interpreted MATLAB Functions for integration of those m-files in an overall Simulink model.

2.2. TIRS Deployment Targets

There are a total of seven different TIRS targets, four of which are used for X-in-the-Loop testing of AFCS. X-in-the-Loop refers to the four target system configurations (model, software, processor and hardware) that are typically used for testing auto-generated code in MBD [25]. AFCS can be defined as a system that uses various sensors in helicopter-like gyros or accelerometers and employs algorithms to provide functionalities ranging from stability and control augmentation to full autopilot modes by deploying an actuation authority over the linkage between the pilot and the swashplate [11].

AFCS Model-in-the-Loop: In MIL testing, the controller model is analyzed along with the simulated plant model. Accordingly, it is the target model into which AFCS and the plant models of TIRS are integrated in the Simulink environment.

AFCS Software-in-the-Loop: In the SIL testing, auto-generated controller code is evaluated. Hence, this is the target model in which auto-generated AFCS s-function and the plant model of TIRS are integrated into the Simulink environment. AFCS C/C++ source code is generated and wrapped within an s-function by using Simulink Coder [32].

AFCS Processor-in-the-Loop: This is the target model in which auto-generated code from the control model is built with all optimization options applied on the compiler and deployed in AFCS target processor, typically to an off-the-shelf evaluation board. It is then integrated to the plant model of TIRS in Simulink in order to conduct non-real-time simulations for testing purposes; thus, non-real-time-related defects can be identified [25].

AFCS Hardware-in-the-Loop: This is the target model in which auto-generated code is deployed to AFCS Hardware. The TIRS plant model is deployed in a real-time target, such as xPC Target, with some I/O and networking facilities as well as auxiliary simulation utilities like management and logging functions. xPC Target is provided by Mathworks as a solution for prototyping, testing and deploying real-time systems using standard PC hardware. It is a bootloader real-time kernel that converts a standard PC into a real-time simulator [34,19].

Conducting experiments with flight simulator is a commonly employed practice to evaluate air vehicles [20,

36,16,2,17]. In addition to quantitative analysis, qualitative ratings can also be collected from the pilots by incorporating flight simulators in the air vehicle design process. Accordingly, the next two targets of TIRS are two flight simulator facilities.

TAI's Engineering Simulator: This is the environment in which Control Loading System, Image Generator and TIRS are integrated. TIRS source code is generated using Simulink Coder Generic Real-time Target (GRT). The GRT is a non-bootloader solution of Mathworks, Inc. and is basically proposed as a way of running real-time simulations under Windows [34].

DLR's Air Vehicle Simulator (AVES): TAI and DLR are collaborating on flight simulator experimentation in AVES. AVES is a modern research flight simulator operated at DLR Braunschweig. It is designed such that interchangeable cockpits of rotorcraft (EC135) and airplanes (A320) can be operated on motion and fixed-base platforms according to particular needs [7]. AVES is the environment in which auto-generated code from TIRS (plant and the controller models) is integrated. TIRS source code is generated using Simulink Coder with 2Simulate Target, which extends GRT in order to generate code that enables seamless integration of air vehicle models in AVES [12].

Desktop helicopter simulation tools are quite popular. Some examples include FLIGHTLAB from Advanced Rotor Craft Technology, Inc. [4] and Heli-Dyn from Aerotim Inc. [13]. The last target of TIRS is a desktop helicopter simulation tool.

TIRS Desktop: This is the environment in which a TIRS executable file and a Graphical User Interface (GUI) are integrated. TIRS source code is generated using Simulink Coder with GRT.

3. AUTOMATING DEPLOYMENT

3.1. Automated Deployment : An overview

Continuous Integration (CI) is a commonly applied practice of software engineering community that mitigates the risks of subsequently finding errors by promoting frequent integration and testing cycles [10]. The burden of integration and testing is minimized through extensive automation. Automated deployment has its roots in CI. It targets at making the deployment activities visible, automating the deployment activities so that they are error free and repeatable [14].

Source control enables the developers to keep the multiple versions of their files, track the changes and revert to any version of a particular file at any time. Source control repositories as the key elements of automated deployment provide a single source for the assets that are subject to deployment. As in software engineering, source control is also commonly applied in MBSSE. As an example, MATLAB/Simulink provides a native integration with Subversion (SVN) [5] and Git [18] source control repositories [27]. This integration is provided via Simulink Project which is a mechanism to manage project files by taking into consideration the version control concerns [31].

The static code analysis facilities are now mostly provided as the integrated capability of development environments. Further automated checks are setup as commit hooks for source control repositories that prevent any code with bad smells to be committed. In MBSSE, static model checking and refactoring is also an emerging practice [26,33,8,1], but their integration into the source control and deployment pipeline is not well established.

Automated testing is also an important aspect of automated deployment. From unit testing to integration testing and system testing, it has been proposed as a field of interest in order to achieve the automated deployment for software intensive systems [14]. The MIL, SIL, PIL and HIL testing practices of MDE brought new challenges to MBSSE. Automation of test case generation and test setup also emerged as challenges in addition to the automated execution of test cases. Some recent efforts that have sought to achieve automated test case generation of simulation models by utilizing model-based approaches [24]. However, further effort is required for an automated test setup which targets deploying simulation models in different configurations to enable X-in-the-Loop testing requirements.

The build scripts can be introduced as another key element of automated deployment. They enable command-line building of large, usually interdependent, source code with appropriate libraries in order to generate executable images. Code generation practices in MBSSE add a further step to the build process. The code generation is considered as an initial step for the build process.

3.2. Causal Block Diagrams

CBD are introduced as a general formalism that is widely utilized for modeling of causal and continuous-time systems [22]. They have been largely employed in control systems design. Denckla and Mosterman refer to the synergy that has been more evident in the last decade among the control systems and software engineering communities, and they introduce CBD as the common language of industry in the embedded systems design process [6].

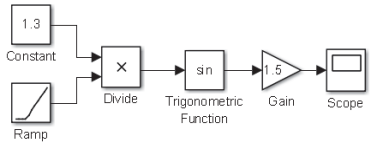


Figure 1 A Causal Block Diagram example

Although there may be other workflow modeling approaches like Business Process Modeling Notation (BPMN)[21] for modeling such automation tasks, and it is also possible to generate automation scripts with code generation, CBD boost the unique modeling capability of the target user group.

While the MATLAB/Simulink implementation of CBD incorporates many further elements with particular semantics, CBD has been basically introduced as graphs made up of connected blocks [35]. A *Block* can be a *Source*, a *Sink* or an *Operation*. While *Source* blocks generate signals, *Sink* blocks consume signals. Operations may be algebraic, trigonometric, logic or, even further algorithmic and these act on the signals. Referring to the example of CBD shown in Figure 1, *Ramp* block is a *Source* block and *Scope* is a *Sink* block. *Sink* is a block that represents a trigonometric operation, and *Gain* is a block that represents an algebraic operation. *Block*s may further possess *Parameters* that configure their operations. The gain *Parameter* of the Gain block specifies the parameter that will be used in product operation. *Blocks* have *InputPorts* and *OutputPorts*, while *Sinks* have only *InputPorts* and *Sources* have only *OutputPorts*. *Operations* have both *InputPorts* and *OutputPorts*. The *Connections* among the blocks represent the signals among the corresponding *OutputPorts* and *InputPorts*.

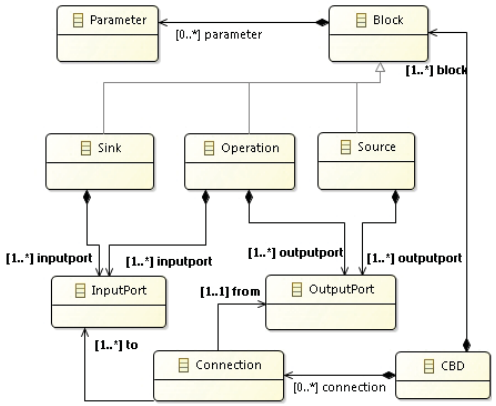


Figure 2 Abstract syntax of Causal Block Diagrams

The abstract syntax of CBD is presented in Figure 2. In addition, Simulink can be declared as the concrete syntax for CBD in this paper.

3.3. Atomic Deployment Blocks

In order to enable automated deployment, this section will propose a set of atomic blocks that are abstracted basic functions. Then, utilizing CBD, these blocks can be used to model a deployment scenario. Thirteen atomic blocks are proposed based on the requirements of the TIRS case. Below, these blocks are explained with their functionalities, inputs, outputs and rationale.

Simulink SVN Checkout: This is the block that will check out a Simulink model from the source control repository to a local folder. The path specification and the file name will be the output of this block. With this block, any Simulink model can be incorporated into the deployment process.

M-file SVN Checkout: This is the block that will check out an m-file from the source control repository to a local folder. The path specification and the file name will be the output of this block. With this block, any m-file can be incorporated into the deployment process.

Simulink Model Analyzer: This is the block that checks a Simulink model for code generation. The user will configure the block for the specific MATLAB/Simulink Model Advisor checks. The execution of a deployment process will be terminated in the event of the failure of any static model checks.

M-file Analyzer: This is the block that checks an m-file for code generation. The execution of a deployment process will be terminated in the event of the failure of any static code checks.

Simulink Code Generator: This block is responsible for code generation using Simulink Coder. The user will configure the block for the appropriate code generation settings. Referring to the TIRS case, it can be GRT or 2Simulate target. The input of the block is the path to the Simulink model to be converted into C/C++ code. Upon completion of a successful code-generation, the paths of the generated source files will be provided as the output of the block

M-file Code Generator: This block is responsible for code generation using MATLAB Coder. The input of the block will come from an m-file block. Upon completion of a successful code-generation, the path to the generated source files will be provided as the output of the block

S-function Generator: This block will use the auto-generated source files to generate S-functions. The input of this block is the paths of generated source files and the output will be the paths to s-functions.

SIL Model Deploy: This block will be used to replace the controller blocks used in the Simulink model with the s-functions obtained from the s-function Generator block for SIL testing. The user will configure this block in order to specify the controller block. The path to Simulink model and the s-function will be the input and the path to modified Simulink model will be the output.

PIL Model Deploy: This block will be used to replace the controller blocks used in the Simulink model with Simulink blocks that enable interfacing with the target evaluation board. The user will configure this block in order to specify the controller block. The Simulink model and the

interfacing Simulink Blocks will be the input and the modified Simulink model will be the output.

PIL Code Deploy: This block is for deploying the code on the target evaluation board. The input of the block will be the source code directory. Further, the user will configure the block with the particulars of the deployment parameters.

HIL Model Deploy: This block is for replacing the controller blocks used in Simulink models with blocks that enable interfacing with the real target hardware. The HIL Model Deploy block will also deploy the model to the real-time target, such as the xPC target. The user will configure this block in order to specify the controller block and to set the particulars of the real-time environment. The Simulink model and the interfacing Simulink Blocks will be the input and the modified Simulink model will be the output.

HIL Code Deploy: This function will be used to deploy the code on the real target hardware. The input of the block will be the source code directory. Further, the user will configure the block with the particulars of the deployment parameters.

Model Deploy: This function is for building an auto-generated code with the specified compiler settings and deploying the generated executable images at a specified location. The input of this block will be the auto-generated code. The compiler and target location will be specified by the user during the configuration of the block.

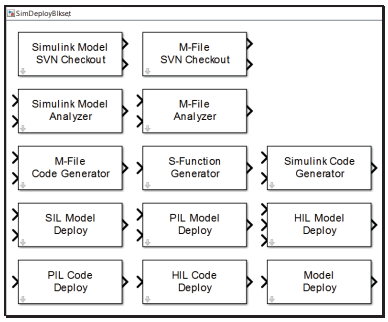


Figure 3 Simulation Deployment Blockset library

4. MATLAB/SIMULINK DEPLOYMENT BLOCKSET

MATLAB/Simulink provides various blocksets which can be accessed through Simulink Library Browser. Modelers also have the capability to create their specific blocksets. Accordingly, the deployment blocks which were introduced previously are designed as a Simulink blockset, known as a "Simulation Deployment Blockset", which is a collection of blocks, as depicted in Figure 3. With this blockset, modelers are provided with atomic functional blocks that they can use to model their overall deployment workflows.

Developing the library model: In order to create a blockset library, the modeler first creates a new *Library* in Simulink. Then, *Subsystem* blocks are added to the *Library* model, in which the modeler constructs the required functionalities by dragging and dropping the components from the Simulink Library Browser. While standard Simulink blocks, such as *inport, outport, constant* and *Matrix Concatenate* can be used, basically parameter, input and output manipulation, most of the functionality that is required for deployment operations is developed using *Interpreted MATLAB Function*, which is a block in which the relation between a single input and a single output port is specified using m-script [28].

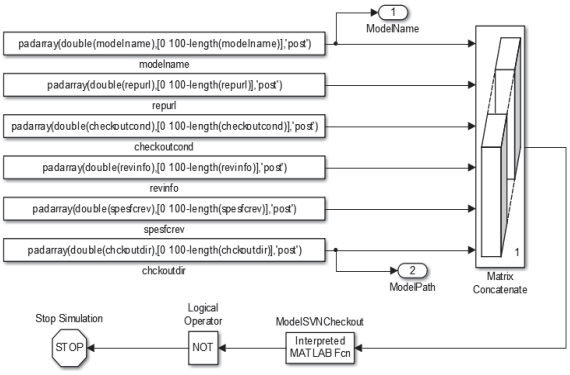


Figure 4 Simulink model SVN checkout subsystem details

The construction of *Simulink Model SVN Checkout* block will be introduced as an example (Figure 4). In this block, the model name, URL of the SVN repository, checkout condition, revision info and checkout directory parameters are specified via *constant* blocks. All the input parameters will be fed into the deployment simulation as *double* values, since, unlike MATLAB, strings are not supported in Simulink. Corresponding ASCII numbers are used to represent strings as arrays of *double* values.

```
function success = ModelSVNCheckout(SVNinfo)
...
%Whole Checkout
if(strmatch(checkoutcond,'on'))
 %Specific Revision Checkout
 if(strmatch(revinfo,'Specific'))
  checkoutstr = sprintf('svn checkout %s -r %s %s',...
   repurl,specfcrev,chckoutdir);
 else %Head Revision Checkout
  checkoutstr = sprintf('svn checkout %s %s',...
   repurl,chckoutdir);
 end
%Single File Checkout
else
...
end
system(checkoutstr);
success = 1;
end
```

Figure 5 An excerpt from Interpreted Matlab Function (Figure 4)

The next step is to perform check out using those input parameters as SVN command line arguments. Then, an

Interpreted MATLAB Function block is added into the subsystem (Figure 5) to implement the SVN command for check out (*svn checkout [-r rev] URL PATH*). The deficiency in supporting multiple input and output ports is overcome by using *Matrix Concatenate* block and *padarray* function to equalize the matrix dimensions. *Interpreted MATLAB Function* block gives the status of checkout as an output argument to show whether the process is successful or not. In case of any failure, deployment simulation is halted.

Upon completion of subsystem, the next step is to develop a simple GUI, known as a "mask", in Simulink.

Developing the block mask: Simulink masks are interfaces to the subsystems that encapsulate the logic inside. Masks basically seal a subsystem by popping up a dialog rather than the model itself and partially hide the model and its components. Mask dialog is a simple GUI component that inhibits a place to display detailed information about the model and fields for the input to the subsystem inside. This provides a layer between the user and the model, thus giving model designers the chance to provide a meaningful icon to the block, define customized parameters whose names reflect the purpose of a block, and provide users with customized documentation that is specific to the masked block [29].

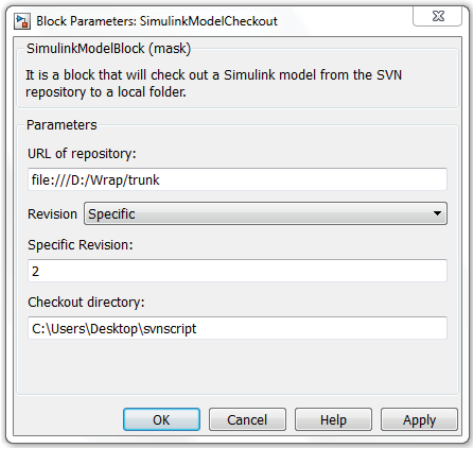


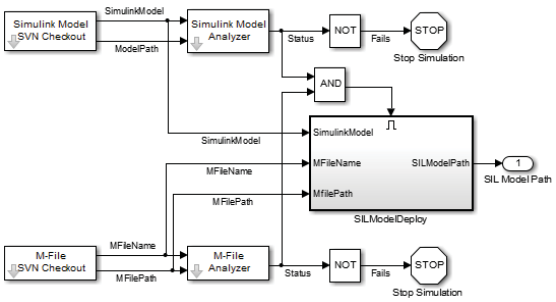
Figure 6 Simulink model SVN checkout block mask

Masks can easily be created using the Mask Editor to wrap a subsystem. There are four panes in a Mask Editor, namely: *Icon&Ports*, *Parameters*, *Initialization* and *Documentation*. The *Icon&Ports* pane allows users to create block icons that can contain descriptive text, state equations, images and graphics. The *Parameters* tab allows users to define dialog box parameter prompts and to name the variables associated with the parameters. The *Initialization* pane allows specifying the initialization commands for mask parameters. And, lastly, the *Documentation* pane is the section in which users add block description and help information for other users to make use of.

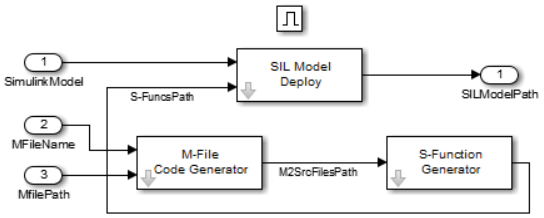
As an example, a *Simulink Model SVN Checkout* block mask is presented in Figure 6. In the Parameters pane, the model name, the URL of the repository, the checkout condition, the revision info and the checkout directory parameters are added. Each parameter is then represented as a GUI component from a variety of options, such as textbox or checkbox.

5. A SAMPLE CASE

The simulation modelers in TAI exercised the Simulink Deployment Blockset with a sample case for a typical deployment of a model SIL testing. They modeled the whole process using Simulink Deployment Blockset, as pictured in Figure 7.



(a) Top level SIL Model Deployment model



(b) Components of a SIL Model Deployment subsystem

Figure 7 Sample model for the SIL Model Deployment process

Deployment process starts by dragging a *Simulink Model SVN Checkout* block from the library browser into a new Simulink model. After filling the necessary fields in the block mask, the block's output ports are fed as input into the *Simulink Model Analyzer* block. This is the block performing the necessary checks on the model to inform the user about model code generation readiness status.

A similar process is carried out for the m-files. Each m-file is checked out and analyzed for code generation using the *M-File SVN Checkout* and the *M-File Analyzer* blocks. If

the model or m-file code generation status fails, then the deployment simulation is halted.

If no process is interrupted, the m-file code generation process starts. M-files are converted into C++ files using *MATLAB Coder*. Then, the result, the source file paths, is sent as input to the *S-Function Generator* block. This is the block wrapping those source files inside an s-function.

Signals carrying the information of s-function paths are then connected to a *SIL Model Deploy* block. This block replaces the *Interpreted MATLAB File* blocks used in the Simulink model with the s-function blocks obtained in the previous step. A modified Simulink model is given as an output from the block.

The sample case improved the required deployment effort for SIL testing in TAI from days to hours. Further, the simulation modelers in TAI find the Simulink Deployment Blockset intuitive. They stated that the accessibility of the automation features in their graphical modeling environment provide them the opportunity to design their deployment scenarios within their modeling workflow.

6. CONCLUSION

Simulation deployment is a collection of activities extending from the development or enhancement of the model to its execution on a particular target. Concerning heterogeneous targets of simulations that are utilized during the development of technical systems, current manual and ad hoc practices are repetitive, labor-intensive, time-consuming and error prone. While the automation of deployment tasks is mostly carried out by the software engineering community using scripting languages, in order to make use of the graphical modeling expertise of the modeling and simulation community, this paper proposes a model-based approach for simulation deployment via employing Causal Block Diagrams.

Following this approach, a Simulink blockset is designed and developed based on the requirements of an industrial case, TAI Indigenous Rotorcraft Simulation. The application is then exercised with a sample case. Then, valuable anecdotal evidences that indicate the success of the approach are collected.

While the modelers are enabled to specify the model deployment process with the graphical modeling methodology which that they are accustomed, this study does not establish the link between the deployment and automated testing. Therefore, future work should include bridging the automated test case generation and execution with the deployment functionality that is proposed in this paper. Tracking the results of tests and relating them to the deployed artifacts, and release management are also parts of future work The ultimate aim is to provide the modeler with support tools and capabilities in their modeling environment so that it is possible to achieve contemporary Continuous Integration advancements in the software engineering community in Model-based Simulation Systems Engineering.

ACKNOWLEDGMENT

The authors would like to thank Holger Duda for careful review and editing. It certainly improved the presentation of this article.

REFERENCES

1. Amelunxen C., Legros E., Schurr A., Sturmer I. Checking and Enforcement of Modeling Guidelines with Graph Transformations. *Applications of Graph Transformations with Industrial Relevance*, Springer (2008), 313-328.
2. Anderson, F., Biezad, D. A Low-Cost Flight Simulation for Rapid Handling Qualities Evaluations during Design. *AIAA Modeling and Simulation Technologies Conference and Exhibit*, Boston, MA, 1998.
3. Balci, O. Guidelines for Successful Simulation Studies. *Proceedings of the 22nd Winter Simulation Conference*, IEEE Press (1990), 25-32.
4. Chengjian, H. *FLIGHTLAB Theory Manual*. Advanced Rotor Craft Technology Inc. Sebastopol, CA, 2004.
5. Collins-Sussman, B., Fitzpatrick, B., Pilato, M. *Version Control with Subversion*. O'Reilly Media, Inc., CA, USA, 2004.
6. Denckla, B., Mosterman, P.J. Formalizing Causal Block Diagrams for Modeling a Class of Hybrid Dynamic Systems. *44th IEEE Conference on Decision and Control*, Seville, Spain, 2005.
7. Duda, H., Gerlach, T., Advani, S. Potter, M. Design of the DLR AVES Research Flight Simulator. *AIAA Modeling and Simulation Technologies Conference*, Boston, MA, 2013.
8. Durak U. Pragmatic Model Transformations for Refactoring in Scilab/Xcos. *Int J Model Simul Sci Comput,* 7: 10.1142/S1793962315410044, 2016.
9. Durak, U., Ören, T. Towards an Ontology of Simulation Systems Engineering, *2016 Spring Simulation Conference*, Pasadena, CA, 2016.
10. Fowler, M., Matthew Foemmel. Continuous Integration. http://www.martinfowler.com/articles/continuousIntegration.html. As of 17 November 2015.
11. Gaul, J.W., Diamond, E.D., Davis, J.M. Automatic Flight Control Systems Development for U.S. Army Heavy Lift Helicopter. *AIAA 12th Aerospace Sciences Meeting,* Washington, DC, 1974.
12. Gerlach, T., Durak, U., Gotschlich, J. Model Integration Workflow for Keeping Models up to Date in a Research Simulator. *Proceedings of the 4th International Conference on Simulation and Modeling Methodologies, Technologies and Applications*, SCITEPRESS (2014), 125-132.

13. Gursoy, G., Tarimci, O., Yavrucuk, I. Helicopter Slung Load Simulations Using Heli-Dyn. *AIAA Modeling and Simulation Technologies Conference*, Minneapolis, MN, 2012.

14. Humble, J., Farley, D. *Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation*. Pearson Education, Upper Saddle River, NJ, USA, 2010.

15. IEEE Std 1730 TM-2010. *IEEE Recommended Practice for Distributed Simulation Engineering and Execution Process (DSEEP)*, 2010.

16. Kiefer, D., Calvert, J. Developmental Evaluation of a Centrifuge Flight Simulator as an Enhanced Maneuverability Flying Qualities Tool. *AIAA Flight Simulation Technologies Conference*, Hilton Head, SC, 1992.

17. Landry, L. Application of Modeling, Simulation and Labs to the F-35 Program. *AIAA Modeling and Simulation Technologies Conference and Exhibit*, Honolulu, Hawaii, 2008. http://dx.doi.org/10.2514/6.2008-6353.

18. Loeliger, J. and McCullough, M. *Version Control with Git: Powerful Tools and Techniques for Collaborative Software Development*. O'Reilly Media, Inc., Sebastopol, CA, 2012.

19. Mosterman, P.J., Prabhu, S., Dowd, A., Glass, J., Erkkinen, T., Kluza, J. and Shenoy, R. Embedded Real-time Control via MATLAB, Simulink, and xPC Target. *Handbook of Networked and Embedded Control Systems*. Birkhäuser Boston (2005), pp. 419-446.

20. Muckler, F. and Obermayer, R. Performance Measurement in Flight Simulation Studies. *AIAA Heterogeneous Combustion Conference*, Palm Beach, FL, 1963.

21. OMG, Business Process Model and Notation (BPMN) Version 2.0.2, Object Management Group, 2013. http://www.omg.org/spec/BPMN/2.0.2/ As of 24 February 2016.

22. Posse, E., De Lara, J. and Vangheluwe, H. Processing Causal Block Diagrams with Graphgrammars in Atom3. *European Joint Conference on Theory and Practice of Software (ETAPS), Workshop on Applied Graph Transformation (AGT)*, Grenoble, France, 2002.

23. Ruff, R., Stephans, C. and Mahapatra, S. Applying Model-Based Design to Large-Scale Systems Development: Modeling, Simulation, Test, & Deployment of a Multirotor Vehicle. *AIAA Modeling and Simulation Technologies Conference*. Minneapolis, MN, 2012.

24. Schmidt, A., Durak, U., Rasch, C., Pawletta, T. Model-Based Testing Approach for MATLAB/Simulink using System Entity Structure and Experimental Frames. *Spring Simulation Multi-Conference*, Alexandria, VA, 2015.

25. Shokry, H., Hinchey, M. Model-Based Verification of Embedded Software. *IEEE Computer*, 42, 4 (2009), 53-59.

26. Sturmer I., Conrad, M., Fey, I., Dorr, H. Experiences with Model and Autocode Reviews in Model-based Software Development *2006 International Workshop on Software Engineering for Automotive Systems*, Shanghai, China, 2006.

27. The MathWorks, Inc. About MathWorks Source Control Integration, http://www.mathworks.com/help/matlab/matlab_prog/about-mathworks-source-control-integration.html. As of 24 February 2016.

28. The MathWorks, Inc. Interpreted MATLAB Function, http://www.mathworks.com/help/simulink/slref/interpretedmatlabfunction.html. As of 24 February 2016.

29. The MathWorks, Inc. Mask Editor Overview, http://www.mathworks.com/help/simulink/gui/mask-editor-overview.html. As of 24 February 2016.

30. The MathWorks, Inc. Model-Based Design. http://www.mathworks.com/solutions/model-based-design/. As of 24 February 2016.

31. The MathWorks, Inc. Simulink in Teams. http://www.mathworks.com/discovery/simulink-projects.html. As of 24 February 2016.

32. The MathWorks, Inc. *Simulink® Coder™ User's Guide*, The MathWorks, Inc., Natick, MA, 2015.

33. Tran Q. M., Wilmes B., Dziobek C. Refactoring of Simulink Diagrams via Composition of Transformation Steps, *The Eighth International Conference on Software Engineering Advances*, Venice, Italy, 2013.

34. Uriarte, F. M., Butler-Purry, K. L. Real-time Simulation Using PC-based Kernels. *Power Systems Conference and Exposition*, Atlanta, GA, 2006.

35. Vagheluwe, H. Causal Block Diagram Algorithms. http://msdl.cs.mcgill.ca/people/hv/teaching/MS/lectures/lecture.CBD/notes.pdf. As of 17 November 2015.

36. van Gool, M., Weingarten, N. Comparison of Low-Speed Handling Qualities in Ground-Based and in-Flight Simulator Tests. *AIAA 1st Flight Test Conference. Flight Test Conference*, Las Vegas, NV, 1981.

B.10 Extending the Knowledge Discovery Metamodel for Architecture-Driven Simulation Modernization

Umut Durak. 2015. Extending the Knowledge Discovery Metamodel for architecture-driven simulation modernization. *Simulation* 91, 12, 1052-1067.

Simulation

Methodology

Simulation: Transactions of the Society for Modeling and Simulation International
2015, Vol. 91(12) 1052–1067

DOI: 10.1177/0037549715615432
sim.sagepub.com
SAGE SCS

Extending the Knowledge Discovery Metamodel for architecture-driven simulation modernization

Umut Durak

Abstract

With the rapid improvement of hardware, the constant evaluation of simulation development methodologies and environments has brought us a new challenge: simulations modernization. This paper presents a simulation knowledge discovery metamodel that extends the Object Management Group's knowledge discovery metamodel by adding a simulation model package for enabling architecture-driven simulation modernization. While there are some endeavors that propose integration methods, modernization of legacy simulations has not been investigated. Architecture-driven modernization has been introduced as a process of comprehending and transforming existing software assets. It advocates a model-based approach to software modernization in which the knowledge extracted from software assets is captured in models that conform to a metamodel, namely the knowledge discovery metamodel. It specifies an ontology of software assets. Model transformations are then recommended as means of modernization of legacy assets. But diversity in methodologies and approaches to specify simulation modeling assets prohibits the Knowledge Discovery Metamodel from providing adequate meta definitions to capture knowledge in simulations. System Entity Structure has long been used for knowledge representation by the simulation community. It provides formalism for composition, taxonomy and coupling relations. Hence, System Entity Structure is proposed as an intermediate metamodel to capture the meta-constructs and their relations. It is then made available to define particular metamodels for particular simulation modeling methodologies or approaches. The promoted methodology is exercised with samples in order to collect evidence of its applicability. Metamodel segments are presented for knowledge discovery for discrete event systems specification and Modelica simulation programming language. A complete metamodel is then introduced for the real-time distributed simulation domain model of the German Aerospace Center (DLR) Institute of Flight Systems. Finally, this metamodel is used to introduce discovery artifacts. These samples provide valuable indications that the methodology works.

Keywords

simulation modernization, Knowledge Discovery Metamodel, System Entity Structure

1. Introduction

1.1. Motivation

The German Aerospace Center (DLR) Institute of Flight Systems (FT) has much experience in building flight simulators.[1–4] The motivation of this effort originated from developing methodologies and tools to modernize legacy flight simulation assets at DLR FT. These assets include code bases from the old Advanced Technologies Testing Aircraft System (ATTAS) and Flying Helicopter Simulator (FHS) ground-based flight simulators.[2,3] They have more than three decades of know-how on modeling and simulation of highly complex aircraft systems under real-time constraints. Therefore, as invaluable assets of the organization, these simulations are being maintained and migrated to the new Air Vehicle Simulator (AVES) facility.[4] Firstly, the approaches to simulation modernization by the modeling and simulation community are investigated.

1.2. Simulation modernization

Legacy simulation is a well-known problem for the community. Over the last 20 years, the simulation industry has

German Aerospace Center (DLR) Institute of Flight Systems, Braunschweig, Germany

Corresponding author:
Umut Durak, DLR Institute of Flight Systems, Lilienthalplatz 7, 38108 Braunschweig, Germany.
Email: umut.durak@dlr.de

utilized various tools and technologies that are now either no longer supported or have considerably lost their market. An example can be illustrated from the real-time simulation of continuous systems. AD 10 was introduced in the early 1980s as a peripheral multiprocessor system, especially designed for simulation of continuous systems. The simulation programming language that supports this architecture was named Modular Programming System 10 (MPS 10).[5] Then, in the late 1980s, AD 100 was introduced with a high-speed floating-point engine and a host controller on a general purpose VAX family. ADSIM was the simulation programming language for the AD 100.[6] Today, these solutions are almost wholly overridden by general purpose real-time operating systems that run over general purpose PCs. Model and software development tasks are blended with contemporary model-based design principles that introduced integration of auto-generated code with real-time simulation frameworks which provide application-specific I/O functions and utilities.[7] Clearly, an enormous amount of knowledge is embedded in the legacy products that have been developed using these technologies.

There has been some effort to tackle legacy simulation problems. Trcka Radosevic et al. define legacy simulations as ''often originating from seventies'' and ''domain-specific, not reusable, complex monoliths.''[8] They claim that, since model development is a time-consuming and expensive activity, methodologies need to be developed for reusing legacy code. They propose a methodology for runtime information exchange between legacy simulations. Likewise, Pullen and White state that ''reuse of legacy code is more effective than trying to re-engineer the code to another language.''[9] They propose wrapping legacy simulations using Java Native Interface and making them available over web services. In 2011, Sonntag et al. defined legacy simulations as ''developed without adhering to known software engineering guidelines, lack of acceptable software ergonomics, run sequentially on single workstation and require tremendous manual task.''[10] Following previous endeavors, they then proposed a service composition-based reengineering of legacy simulations. Their solution is also grounded in the idea to build a Java wrapper around legacy simulations to make them available as web services.

Lacy attacks incompatible model representation of legacy discrete-event simulation tools.[11] He notes that each tool uses different concepts and approaches for representing process interactions and proposes the Process Interaction Modeling Ontology for Discrete Event Simulations (PIMODES) for a unified model representation for process-oriented discrete event simulations in order to enable model interchange. Whilst this effort does not particularly target modernization, it reports promising results as to how metamodeling and ontologies can help to extract knowledge from model specifications of legacy discrete event modeling tools.

It is quite interesting that, although most of the previous studies state that legacy simulations lack current software engineering qualities, they rather choose to keep them static and integrate them into current systems, complete with their deficiencies, rather than proposing a simulation modernization methodology to make legacy simulations better. On the other hand, the software engineering community has long been working on modernization methodologies for legacy software.

1.3. Software modernization

The early studies on software modernization can be traced back to Lehman's work in the 1980s.[12,13] Lehman's ideas, later named as the laws of software evolution, state that software systems which are actively used and embedded in a real-world domain are subject to an inevitable change and evolution.[13]

With evolution, due to constant maintenance, software systems encounter software aging and erosion. Sourceless executables, dead data, dead code, inconsistencies, and missing capacities provide some metrics to measure the aging and erosion.[14] Parnas relates software aging to two distinct causes:[15] the first is not making the required changes and, ironically, the second is making changes. The first, he terms as ''lack of movement'' and the second as ''ignorant surgery.'' On the one hand, software systems age due to changes in technology, user expectations and environments, and, on the other, due to successive changes with limited understanding of the original or previous design concepts and decisions. Both result in software systems becoming less maintainable.[16] They eventually degrade in performance, become buggy and lose their customers and market.[15]

Since wholescale replacement of existing systems is risky and not economical, reengineering is introduced for software systems to tackle aging and erosion. Chikofsky and Cross defined reengineering as the examination and alteration of the existing system with respect to new requirements that could not be met.[17] Thus, reengineering focuses not on discarding the system as whole, but preserving the knowledge embedded in the legacy system by proposing an evolutionary maintenance of the legacy systems in order to reduce risk and cost.[18]

Efforts to define a standard reengineering process for software systems have led to Architecture-Driven Modernization (ADM). This was introduced by the Object Management Group (OMG) as a standard approach for the modernization of existing systems with a broad focus on all aspects of the current system and a promise of transformation to target architectures.[19] It advocates a model-based approach to software modernization in which the knowledge extracted from software assets is captured in models that conform to a metamodel, namely the Knowledge Discovery Metamodel (KDM).[20] Model

transformations are then recommended as a means of the modernization of legacy assets.

In terms of modernizing legacy simulation assets, with its promise to transform embedded implicit knowledge in the legacy assets to target architectures, the ADM approach is regarded as promising. However, there is a wide diversity of methodologies and approaches to specify simulation modeling assets. Silver et al. presented a not definitive but solid review of taxonomies for simulation modeling methodologies and approaches.[21] They basically differ depending on the modeled system behavior (e.g. continuous, discrete, hybrid), focus of the modeler (e.g. activity diagrams, state transition diagrams), abstraction (e.g. agent-based simulation, object-oriented simulation), execution (e.g. activity scanning, event scanning) or model syntax (e.g. declarative, functional). While there are well-established formalisms for representing software assets, they are thus captured in KDM as an ontology, the diversity in methodologies and approaches to specify simulation modeling assets prohibits KDM providing adequate meta definitions to capture knowledge in simulations.

1.4. Overview of approach

For adopting ADM for modernizing legacy simulation assets, it is claimed that KDM requires a particular extension. This effort extends KDM by adding a simulation model. This extended KDM is called the Simulation Knowledge Discovery Metamodel (SKDM). It employs System Entity Structure (SES), which has long been used for knowledge representation by the simulation community. SES provides formalism for composition, taxonomy and coupling relations.[22] Hence, it is proposed as an intermediate metamodel to capture the meta simulation constructs with their relations. As the metamodeling approach for a particular simulation modeling methodology or application, metamodel developers are encouraged to construct their specific simulation knowledge discovery metamodel upon the proposed intermediate metamodel provided in SKDM.

Model-based approaches and metamodeling for the forward engineering of simulation systems has long been studied. In 2002, Tolk introduced a discussion to apply model-based approaches to simulation development.[23] Some of the notable studies include those by Durak et al.,[24] Topçu et al.,[25] and Cetinkaya et al.[26] Nevertheless, the link between these endeavors and legacy simulation literature has never been established. This paper presents an attempt to address this link. It concentrates on a metamodeling for simulation knowledge discovery in simulation assets. Thus, it attempts a contribution towards enabling OMG's model-driven modernization approach for modeling and simulation.

1.5. Scope

The paper presents the proposed extension to the KDM and the approach to develop metamodels for knowledge discovery in simulation assets using this extension. It will begin by introducing a background to ADM, KDM, and SES. Then, it will present SKDM and its utilization. Later, it will provide three sample cases to exemplify the employment of SKDM for three different simulation modeling approaches with particular motivations. The first two examples are constructed particularly to exercise metamodeling. The first sample case will exploit the construction of a metamodel for a discrete simulation model using SKDM. In the second sample case, SKDM will be employed to construct a metamodel for knowledge discovery from declarative, object-oriented simulation models in the Modelica simulation programming language.[27]

The particular approach to design real-time distributed simulations at DLR FT has evolved to a simulation framework called 2Simulate, which, in particular, implements an institute-wide shared domain model.[28] In the last example, a complete metamodel will be presented for this real-time distributed simulation domain model. Then, this metamodel will be used to introduce sample discovery artifacts from a legacy helicopter simulator asset.

2. Background

2.1. Architecture-Driven Modernization

The OMG ADM Task Forces define ADM as ''the process of understanding and evolving existing software assets'' and sets the most important point of the task as a standard metamodel that will be used to represent existing software assets.[29]

The horseshoe model for software reengineering was introduced by the Software Engineering Institute in the late 1990s. It defines three processes: the first process, analysis of an existing system, goes up the left leg; the second process, logical transformation, goes across the top of the horseshoe and the last, development of the new one, goes down the right leg of the horseshoe.[30] These processes are also referred to reverse engineering, restructuring and forward engineering. This metaphor was adopted by OMG ADM Task Forces by introducing three levels that represent the level of abstraction that is reached during modernization: technical modernization, application/data modernization and business modernization.[31] Technical modernization is defined as the most common modernization effort that targets mostly language or platform change. Application/data modernization is described as targeting a change in systems design. Lastly, business modernization is presented as the one that addresses the business rules and processes that are governed by the software. The OMG ADM Task Forces claimed that any modernization

effort, no matter at which abstraction level it is, follows the horseshoe model, but requires different tools.

ADM proposes recovering the artifacts as models at any particular abstraction level that the modernization effort requires and then employing model transformations. In reverse engineering, the implicit knowledge embodied in software assets is recovered and represented in models according to certain metamodels. KDM can be introduced as one of the metamodels that OMG proposed for technical modernization. Then, various restructuring and refactoring methods are utilized over these models, including transformations to a target model. Then, well-known forward engineering methodologies are employed to regenerate modernized software assets.[32]

2.2. Knowledge Discovery Metamodel

Reverse engineering requires a complete overhaul of available assets to understand the diverse aspects of the available knowledge. Thus, the reverse engineering of software systems is sometimes referred to as software archeology.[33] As the archeology requires understanding the civilizing and cultural forces that create available artifacts, reverse engineering requires a complete analysis of available artifacts, such as source code, databases, user interfaces, or repositories, as well as the methodologies and infrastructures.[32] As a facilitator, KDM proposes a standard and integrated way to represent all software artifacts in a certain legacy system.

KDM defines a metamodel to represent existing software, its elements, associations and operational environments, and covers a large and diverse set of applications, platforms and programming languages.[20] The specification is organized in four layers: Infrastructure Layer, Program Elements Layer, Runtime Resource Layer, and Abstractions Layer. As presented in Figure 1, each of them is based on the previous one. These layers are further organized to packages, each of which corresponds to a certain independent facet of knowledge about the software. These packages define KDM models.

The Infrastructure Layer specifies the fundamental metamodel element types and constraints. It comprises the Core, ''kdm'', and Source packages. The Core package provides a set of base types that are used to derive any metamodel element.

The ''kdm'' package provides the metamodel elements, classes and associations to specify the structure of KDM instances and, thus, defines the organization of KDM. The Core and ''kdm'' packages are not used for a specific model, but utilized as the foundation of KDM.

The Source package consists of elements to present tangible artifacts of the existing system, e.g. source files or resource descriptions. It is used to construct an inventory model that targets capturing the knowledge about what artifacts exist in the system, the role of each artifact (source file, executable or configuration description), the organization of artifacts (directory structure) and dependencies between artifacts.

The Program Elements Layer has Code and Action packages. They define metamodel elements to provide a language independent intermediate representation of common programming language constructs. Using these two packages, a Code model, which represents the implementation level assets of the existing software system, can be constructed. While the Code package is used to represent name elements from the source code, the Action package is used to represent behavioral aspects such as control flows.

The Runtime Resources Layer has elements to describe the operating environment of the existing software system. These elements are provided in Platform, UI, Event, and Data packages and corresponding models. The Platform package has metamodel elements to present the runtime environment, like CORBA. Sample concerns of the Platform metamodel are: what elements of the run-time environment are used or what are the bindings to the runtime environment. The UI package provides metamodel elements to represent information related to the user interface: What data originate from the user interface? or What is the organization of the user interface? The Event package introduces metamodel elements for representing the high level behavior of the application. The states, state transitions and events are some of the elements of this

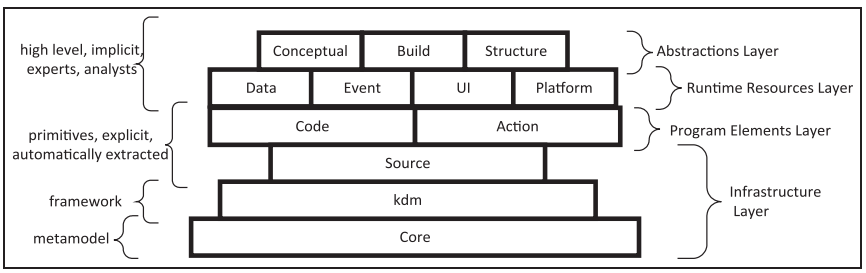


Figure 1. The structure of KDM packages.[20]

package. The last package of the run-time resources layer is the Data package. It defines metamodel elements to represent the structure of the persistent data in the existing software system.

The Abstractions Layer possesses Structure, Conceptual, and Build packages and, thus, enables it to represent domain specific or application specific abstractions. It also provides metamodel elements to recover information about the build process. Subsystems, layers, packages and their organization in an existing software system are captured by the metamodel elements that are introduced in the structure package. The Conceptual package includes metamodel elements to build a conceptual model for the reengineering effort. Thus, domain vocabulary, scenarios and business rules can be captured. The information about the build process of the existing system is recovered using the metamodel elements defined in the Build package. These metamodel elements include build tools, build steps, libraries etc.

KDM is well applicable for knowledge discovery in simulation software. The Action package, for example, can be employed for knowledge discovery of behavior, or the UI of a simulation software can be reversed engineered utilizing the UI package. However, the knowledge discoverable with the available KDM packages will barely encompass anything about simulation modeling; rather it will regard simulation as software. As KDM is presented as the ontology for software assets, Silver et al. attempted to develop an overall ontology of discrete event modeling and simulation,[21] the discrete-event modeling ontology (DeMO). They specify four different types of a discrete event model: process oriented model, activity oriented model, state oriented model, and event oriented model. Each has various formally defined offsprings with their own properties and property restrictions. Hence, DeMO can be used to discover the knowledge about the type of model in a discrete event simulation, as well as the properties of the model. Durak et al. illustrated how trajectory simulation is structured in their Trajectory Simulation ONTology (TSONT).[34] The constructs of trajectory simulations, including various models, like aerodynamic or sensor models, their hierarchies, their parameters and interfaces, as well as their composition, are all captured in TSONT. Thereby, it can be employed as a metamodel to extract knowledge from a trajectory simulation about the type of models used, the parameters utilized and how they are composed. Further examples can also be listed, but the idea is that modeling and simulation knowledge discovery requires further modeling and simulation methodology or application-specific metamodels. Therefore, the simulation model package, which is based on the SES, is proposed to enhance the KDM for simulation knowledge discovery. The particulars of SES are introduced in the following section.

2.3. System entity structure

SES is defined as a schema for knowledge representation of decomposition, taxonomy and coupling of systems.[22] The system theory-based approach to modeling and simulation has resulted in many enhancements in the field, one of which is SES.[35] It enables compact specification of a family of models utilizing interactions of decomposition, coupling and taxonomies.[36] SES is also defined as a formal ontology framework to specify the elements of a system and their relations in hierarchical fashion.[37]

SES nodes and relationships, and an example SES,[38] are given in Figure 2. A labeled tree is used to represent an SES. It possesses four types of nodes: Entity, Aspect, Specialization, and Multi-Aspect. The Entity node has been described as a real-world object that corresponds to a model component. Variables can be attached to entities. Aspect, on the other hand, describes one decomposition of an entity out of many possibilities. Thus, children of Aspect nodes are entities which connect to the parent node in one possible decomposition aspect. Aspects are represented by vertical lines. The children of Specialization are variants of their parents. Parents are categorized into children using a specialization node.[39] The representation of Specialization is double line arrows. When an entity consists of a collection of homogeneous components, it is called multiple entity and specified using Multi-Aspect.[22] Three vertical lines are used to represent Multi-Aspects.

The SES is characterized by a set of axioms: *uniformity, strict hierarchy, alternating mode, valid brothers, attached variables*, and *inheritance*.[40] *Uniformity* indicates that any two nodes with the same labels have isomorphic subtrees. *Strict hierarchy* prevents a label from appearing more than once down any path of the tree. *Alternating mode* says that, if a node is entity, then the successor is either aspect or specialization, and vice versa. Having two brothers with the same label is prohibited by *valid brothers. Attached variables* state that variable types attached to the same item shall have distinct names. With *inheritance*, it is stated that specialization inherits all variables and aspects.

SES is proposed as the basis of the proposed intermediate metamodel in the Simulation Model package, in consideration of, on one hand, its roots in the theory of modeling and simulation, but on the other, its expressive power and clarity with a small number of axioms. Hence, SES suits as a simple intermediate metamodel which dictates a formal structure that is easy and accessible.

3. Simulation knowledge discovery metamodel

While KDM provides a complete ontology for software assets, it has no particular architectural viewpoint that supports recovering information from simulation models. For

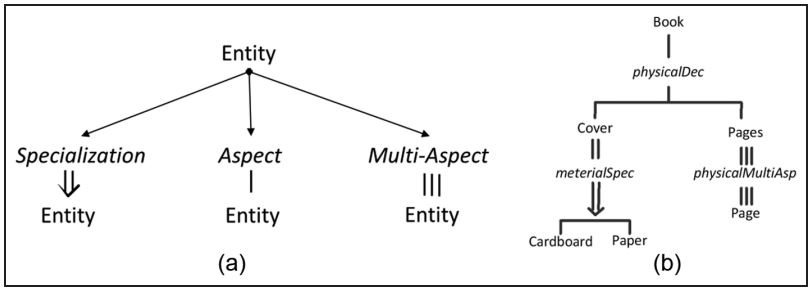


Figure 2. (a) SES nodes and relationships,[37] and (b) an example.[38]

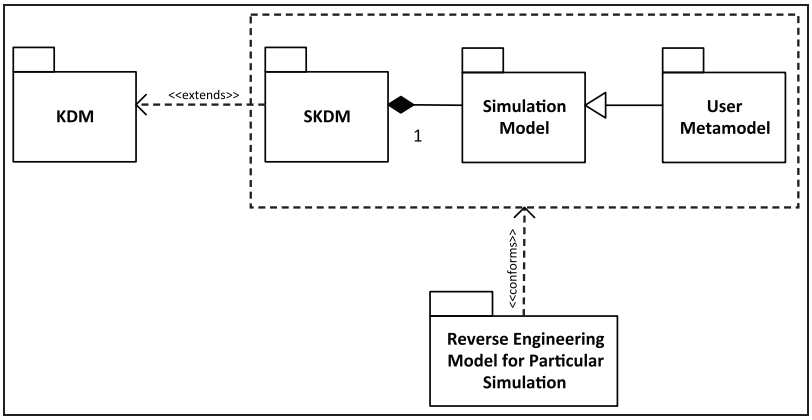


Figure 3. Simulation Knowledge Discovery Metamodel in use.

extending KDM to support simulation knowledge discovery, a Simulation Model package is designed in the Runtime Resources Layer. This extended KDM is termed SKDM.

As shown in Figure 3, SKDM extends KDM with a Simulation Model package. However, the metamodel developer needs to understand that SKDM is not complete. Like the Core package, it cannot be used for modeling, but rather as a fundamental package based on SES that provides an abstract metamodel used to derive concrete metamodel elements while metamodeling for a specific simulation methodology or approach. For the metamodel of a particular simulation modeling methodology or approach, the metamodel developer needs to define their own concrete metamodel elements for the user metamodel via inheriting them from abstract metamodel elements of the Simulation Model.

In the following paragraphs, the Simulation Model package will be introduced following the structure and pattern of other packages from KDM.[20]

This package provides a set of metamodel elements for describing metamodel classes for creating a package of a particular simulation modeling methodology or approach. Using SES, simulation model package classes provide a hierarchical structure to represent taxonomies and couplings.

The Simulation Model package consists of two class diagrams:

- Simulation Model;
- Simulation Model Inheritance.

The Simulation Model class diagram extends the KDM framework with specific metamodel elements of SES. The class diagram of these classes and their relations are shown in Figure 4.

The Simulation Model is the specific KDM model that corresponds to the simulation model of the existing system. While all other elements in the Simulation Model package are abstract, all concrete classes from these abstract elements will be owned by the Simulation Model. It provides a container for various simulation model elements. Therefore, the implementer is recommended to arrange simulation model elements into one or more Simulation Model containers.

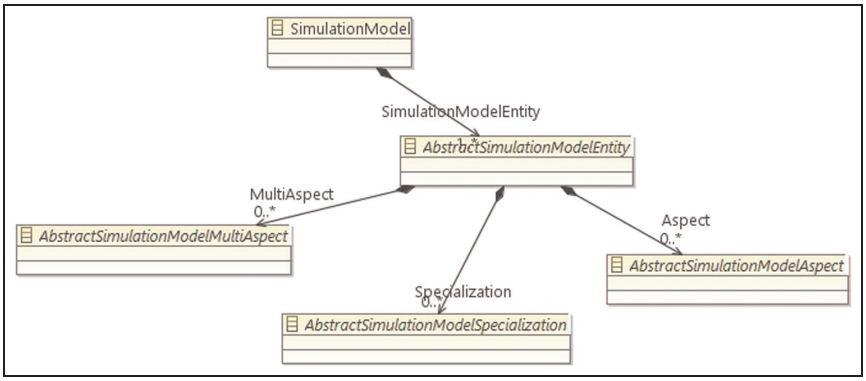


Figure 4. Simulation Model class diagram.

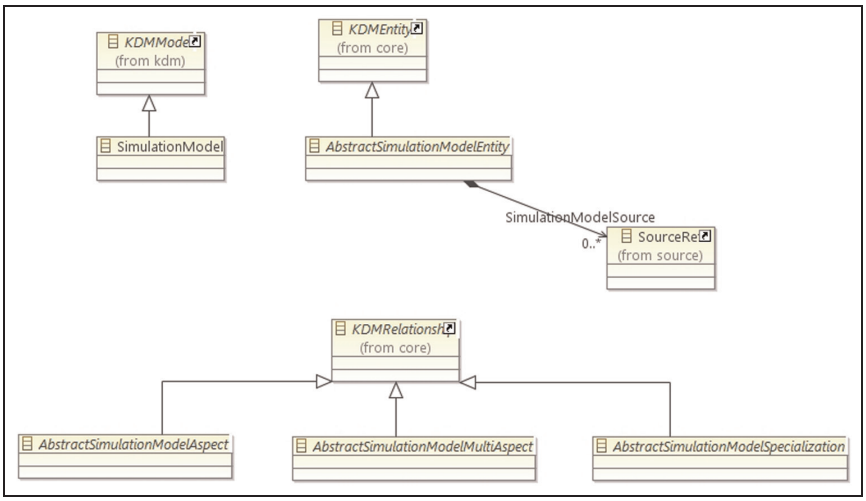


Figure 5. Simulation Model Inheritance class diagram.

AbstractSimulationModelEntity is an abstract superclass of concrete simulation model entities. It represents the Entity node in SES and is derived from KDMEntity. It has zero or more concrete aspect, multi-aspect, and specialization relationships that are derived from AbstractSimulationModelAspect, AbstractSimulationModelMultiAspect, and AbstractSimulationModelSpecialization, respectively. The implementer should map specific simulation entities into concrete subclasses of AbstractSimulationModelEntity that will be defined in the particular corresponding packages.

AbstractSimulationModelAspect is an abstract superclass for concrete aspect relationships. It has its semantic roots in SES and represents the Aspect node in SES. AbstractSimulationModelSpecialization is an abstract superclass for concrete specialization relationships. It is the representation of the Specialization node from SES. AbstractSimulationModelMultiAspect is an abstract superclass for concrete multi-aspect relationships and is based on the definition of Multi-Aspect node in SES. These three classes are derived from KDMRelationship. The implementer is recommended to extend these classes to specify concrete aspect, multi-aspect, and specialization relationships in a particular corresponding package.

The Simulation Model Inheritance class diagram introduces how the classes of the Simulation Model package are related to the metamodel elements defined in the Core package. The classes and associations are shown in Figure 5.

4. Sample cases

After illustrating how KDM is extended with a Simulation Model package that defines the abstract superclass which

is adopted from SES, three sample cases will be presented in this section. The three samples are selected in such a way that they all have distinct approaches to simulation modeling. The first sample adopts an approach from the DEVS community, while the second one comes from the Modelica community. These first two sample cases exemplify only metamodeling. They exploit how a simulation modeling approach can be captured as KDM-based on the classes in the SKDM simulation model package. They are representative portions rather than complete metamodels. The third sample, on the other hand, presents a complete metamodel for an in-house simulation domain model. It will present a metamodel that adopts DLR FT's real-time distributed simulation domain model and further exemplifies knowledge discovery from a legacy simulation software using this metamodel.

4.1. DEVS simulation model discovery metamodel

This sample case adopts DeMO, which was presented as a simulation ontology that captures classical discrete event simulation world views, various formalisms, and modeling techniques that conform to the world views. Silver et al. state that the meaning of concepts used in simulation software is encoded and cannot be explicitly shared with other systems or humans.[21] DeMO attempts to tackle this problem by describing discrete event simulation concepts, and the relations among them, explicitly in a form that is both human and machine readable.

DeModel is introduced as the top level concept of the taxonomy for discrete event modeling. DeModel has subclasses that corresponds to a particular discrete event modeling formalism, e.g. a process-oriented model or activity-oriented model. Every DeModel has a mechanism which defines how the components operate and has components which are the building blocks of the model. Transition function and clock function can be declared to be sample components. Likewise, event scheduling and transition triggering can be given as sample model mechanisms. While the DeMO ontology covers considerably more than what has been mentioned in this paragraph, as a sample case, a DeMO package was created using just the information provided in this paragraph. So, it is neither exclusive nor complete. Below is the presentation of the DeMO package following the pattern and structure of KDM.[20]

A DeMO package defines a set of metamodel elements to represent discrete event simulation concepts and relations. Using these metamodel elements, information about the simulation components, mechanisms can be recovered from the existing discrete event simulation software. The concerns of the DeMO package are as follows:

- What are the components of the existing discrete event simulation?
- Which mechanisms are used to operate these components?

The DeMO view can be constructed by an expert via analyzing the code. If there is a simulation tool or infrastructure in use, an extractor tool can also be developed that utilizes infrastructure API and semantics for the given infrastructure to produce DeMO views. Alternatively, if there is a language in which the elements of discrete event simulation are explicitly defined, parsers can be developed to produce DeMO views.

The organization of the DeMO package consists of two class diagrams:

- DeMO Model;
- DeMO Inheritance.

The DeMO Model class diagram shows the classes of the DeMO package and their relations, as shown in Figure 6.

DeModel is a simulation model entity that represents a discrete event model. It is a subclass of Abstract SimulationModelEntity. It has two aspects, which are connected with it via ModelAspect, namely ModelMechanism and ModelComponents. Furthermore, two variants of DeModel are defined using ModelSpecialization. These are the ProcessOrientedModel and the Activity OrientedModel. ModelMechanism has also two variants which are defined using MechanismSpecialization, namely EventScheduling and TransitionTrigering. Model Components is a set of ModelComponent. This relation is specified using ComponentMultiAspect. And, lastly, ModelComponent has two variants: TransitionFunction and ClockFunction. ComponentSpecialization class specifies this specialization.

The DeMO Inheritance class diagram shows the inheritance of the metamodel elements of the DeMO package, as seen in Figure 7.

4.2. Modelica simulation model discovery metamodel

Modelica was introduced as a domain-specific language for physical systems modeling which is built on acausal modeling with mathematical equations in an object-oriented manner.[41] It is widely used in mechatronics, automotive, and aerospace applications which require modeling of complex and heterogeneous systems with mechanical, electrical, and hydraulic subsystems. As the second sample case, SKDM will be extended with a Modelica package. This sample case will exemplify how a simulation modeling package can be used to construct a metamodel for a simulation modeling language.

The Modelica package will adopt a SysML4Modelica profile that has been developed in an OMG attempt at

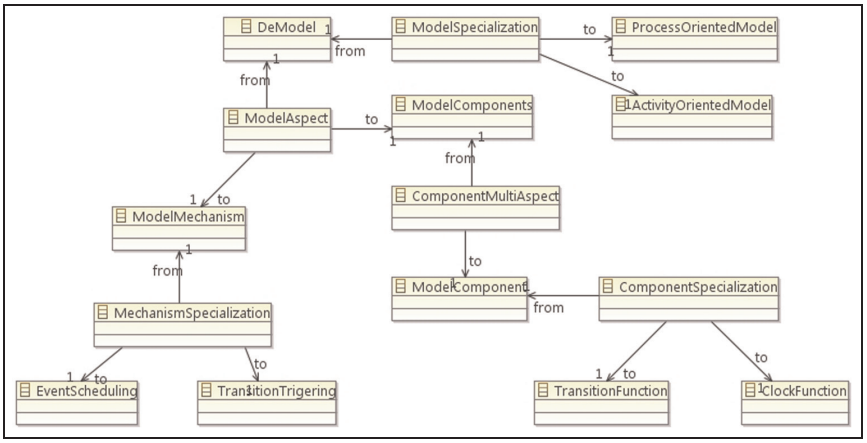


Figure 6. DeMO Model class diagram.

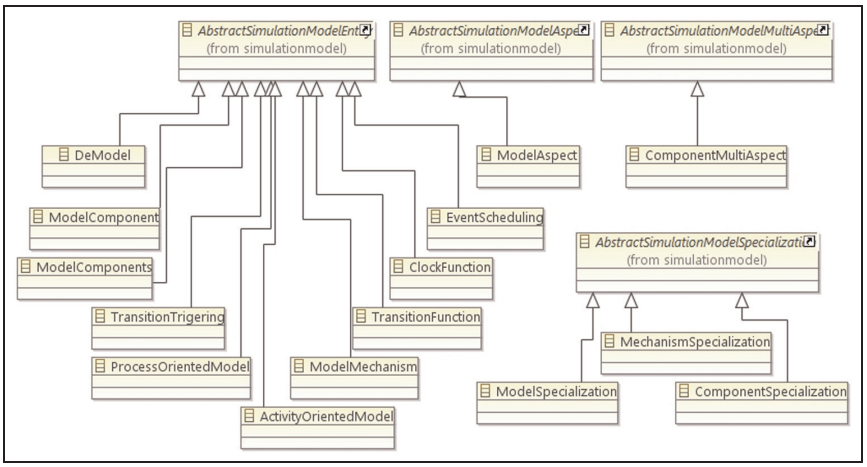


Figure 7. DeMO Inheritance class diagram.

developing a SysML-Modelica transformation specification.[42] The intended use case of a Modelica package can be stated as recovering information from legacy Modelica models. It is described using a Simulation Model package of SKDM. A simple partial excerpt of this profile, rather than a full one, is utilized to create an example metamodel which describes only the top level Modelica constructs. Below is the presentation of the Modelica package following the pattern and structure of KDM.[20]

A Modelica package defines a set of metamodel elements to represent Modelica simulation modeling language constructs. These metamodel elements can be utilized to extract information about Modelica language constructs used in an existing Modelica simulation. The concern of the Modelica package is to identify which of the Modelica language constructs are used. To develop Modelica views automatically, it is possible to develop a parser that will analyze Modelica code to extract information.

The organization of the Modelica package consists of two class diagrams:

- Modelica Model;
- Modelica Inheritance.

The Modelica Model class diagram shows the classes of the Modelica package and their relations, as seen in Figure 8.

ModelicaClassDefinition defines the basic structural unit in Modelica. It is a subclass of AbstractSimulationModelEntity and an abstract superclass for all Modelica constructs. typeOf is derived from AbstractSimulationModelSpecialization to specify the

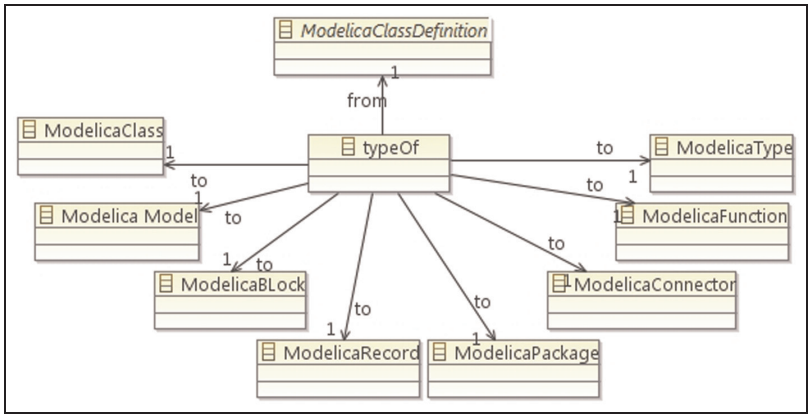


Figure 8. Modelica Model class diagram.

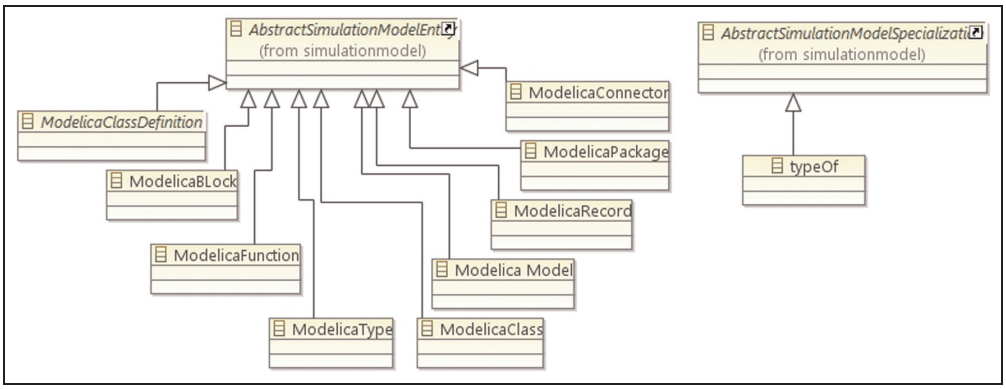


Figure 9. Modelica Inheritance class diagram.

variants of the Modelica constructs that we are interested in. For Modelica, ModelicaClass and ModelicaModel both have the same semantics, and are the most general specialized classes. ModelicaBlock is quite similar to ModelicaModel, but all its connectors are either input or output. ModelicaRecord is specified as a specialized Modelica class definition that is allowed to contain only public declarations. ModelicaPackage is another specialized Modelica class definition that contains the declarations of classes and constants. ModelicaConnector is a construct that is restricted not to contain equations and algorithms and ModelicaFunction is a specialized class definition for a section of procedural algorithmic code. Lastly, ModelicaType is a construct that is restricted to predefined types, enumerations or arrays.

The Modelica Inheritance class diagram shows the inheritance of metamodel elements of the Modelica package, as seen in Figure 9.

4.3. *Model discovery with real-time distributed simulation domain model*

At DLR FT, over the years of developing flight simulators and configuring them for various projects, the approach to design real-time distributed simulations has been captured as a simulation framework called 2Simulate.[28] 2Simulate acts as the overall simulation framework for simulation development endeavors at DLR FT, and also implicitly possesses a shared real-time distributed simulation domain model. This implicit model is specified as a metamodel by adding 2Simulate package to SKDM. Thus, using this metamodel, knowledge discovery from the legacy simulation assets became possible. Below is the presentation of a 2Simulate package following the pattern and structure of KDM.[20]

A 2Simulate package specifies the metamodel elements to present DLR FT real-time distributed simulation

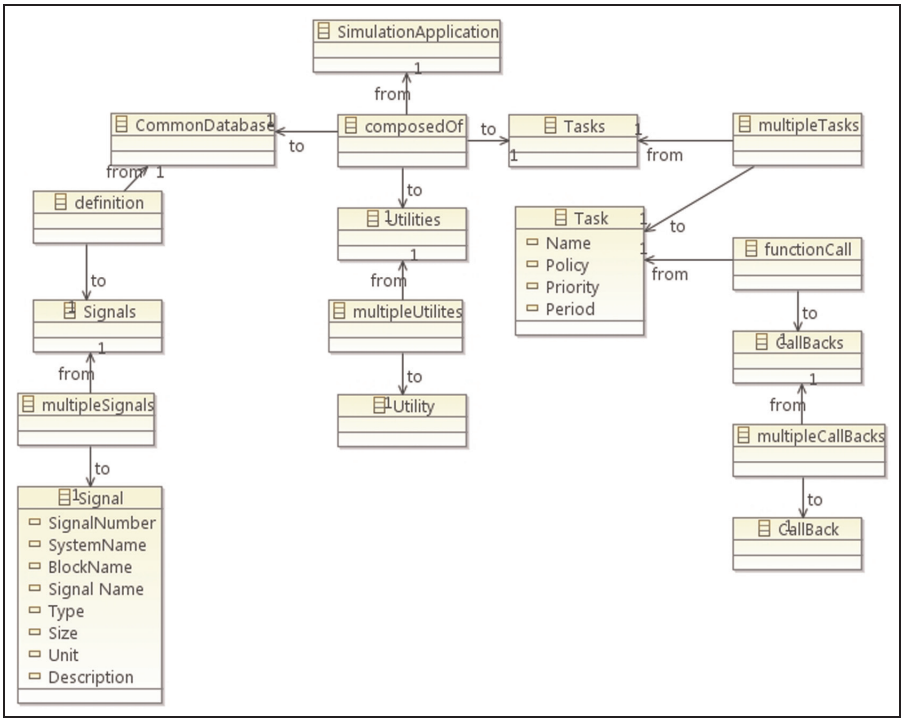


Figure 10. 2Simulate Model class diagram.

concepts and their relations. This metamodel aims at enabling its users to recover information about how these concepts are utilized in a simulation software asset. The concerns of a 2Simulate package are as follows:

- Which real-time simulation constructs are used in existing simulation software?
- How are these constructs configured for the existing simulation software?

The 2Simulate view can be constructed by an expert via analyzing the code. It is also possible to develop an extractor tool to analyze the code for 2Simulate API to produce 2Simulate views.

The organization of a 2Simulate package consists of five class diagrams:

- 2Simulate Model;
- 2Simulate Inheritance;
- Task Hierarchy;
- Callback Hierarchy;
- Utility Hierarchy.

The 2Simulate Model class diagram presents the top level classes of the 2Simulate package and their relations, as shown in Figure 10.

2Simulate supports any simulation application that is composed of real-time tasks, some simulation utilities and a common database that specifies the data exchange among the distributed elements. This is captured by SimulationApplication, tasks, CommonDatabase, and Utilities that are derived from AbstractSimulation ModelEntity. composedOf, on the other hand, is derived from AbstractSimulationModelAspect. Tasks and Utilities are related to a single Task or a Utility using multipleTask and multipleUtility classes that derive from Abstract SimulationModelMultipleAspect. The functionCall aspect relates task to CallBacks. Likewise, the definition aspect relates the signals to CommonDatabase. And, again, multi-aspect classes are used to relate signal to signals and CallBack to CallBacks. The classification hierarchy of CallBack and task is given in Figure 11.

There are five types of CallBack. These are PreInitCallBack and PostInitCallBack, which are fired just before and after the initialization of a task, and PreProcessCallBack, IntermediateProcessCallBack and PostProcessCallBack, which are fired before, during and after each execution of a task. callbackTypes is derived from AbstractSimulationModelSpecialization in order to define these variants.

Figure 12 presents the hierarchy for the variants of a task. taskTypes, modelTaskType, and ioTaskTypes are the

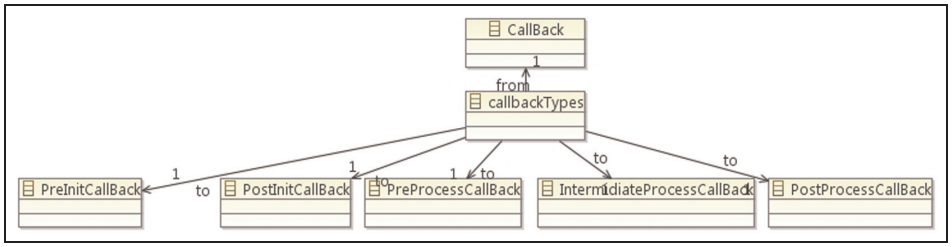


Figure 11. CallBack Hierarchy class diagram.

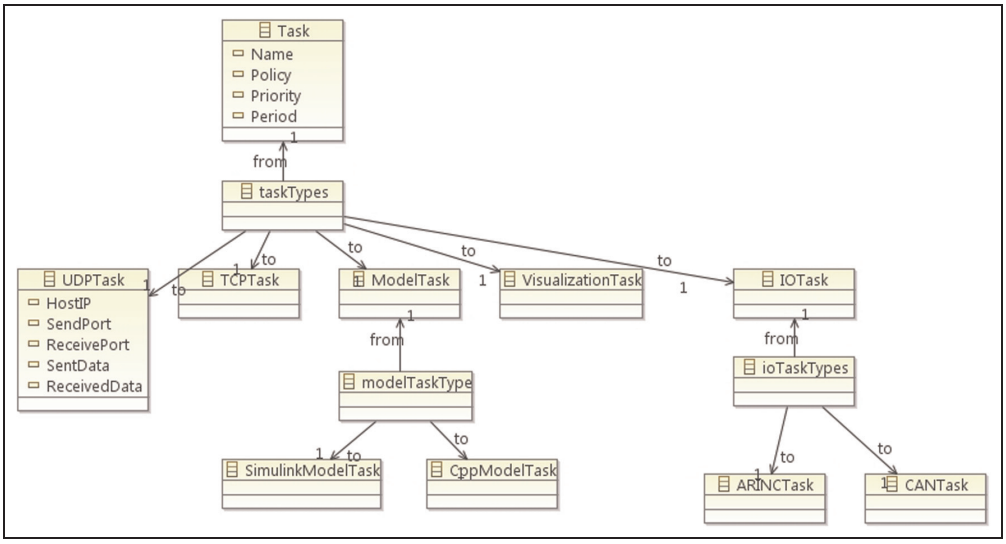


Figure 12. Task Hierarchy class diagram.

classes that are derived from Abstract SimulationModelSpecialization to present the hierarchy of tasks. Top level variants of task are UDPTask for UDP communication, TCPTask for TCP communication, ModelTask for dynamic models, VisualizationTask for 3D visualization, and IOTask for device input–output (IO). Then, ModelTask also has its variant for Simulink ModelTask for Simulink models and CppModelTask for dynamic models developed using C++ . On the other hand, IOTask has its variants according to device IO types, ARINCTask for communication ARINC devices and CANTask for communication CAN devices.

Utilities are defined as real-time distributed simulation services. Two Utility variants that are captured by the 2Simulate package are DisplayUtility and MonitorUtility. DisplayUtility refers to a service that corresponds to displaying signals in any particular way during runtime, while MonitorUtility corresponds to monitoring the values of signals. As illustrated in Figure 13, utilityTypes which derive from AbstractSimulationModelSpecialization, is used to specify these variants.

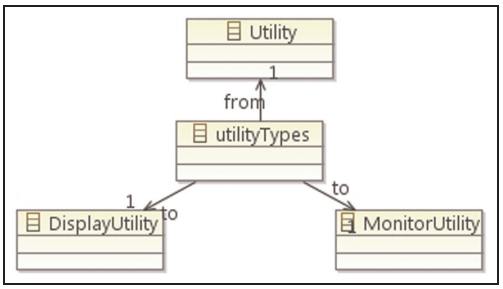


Figure 13. Utility Hierarchy class diagram.

The 2Simulate Inheritance class diagram shows the inheritance of top level metamodel elements of the 2Simulate package, as seen in Figure 14.

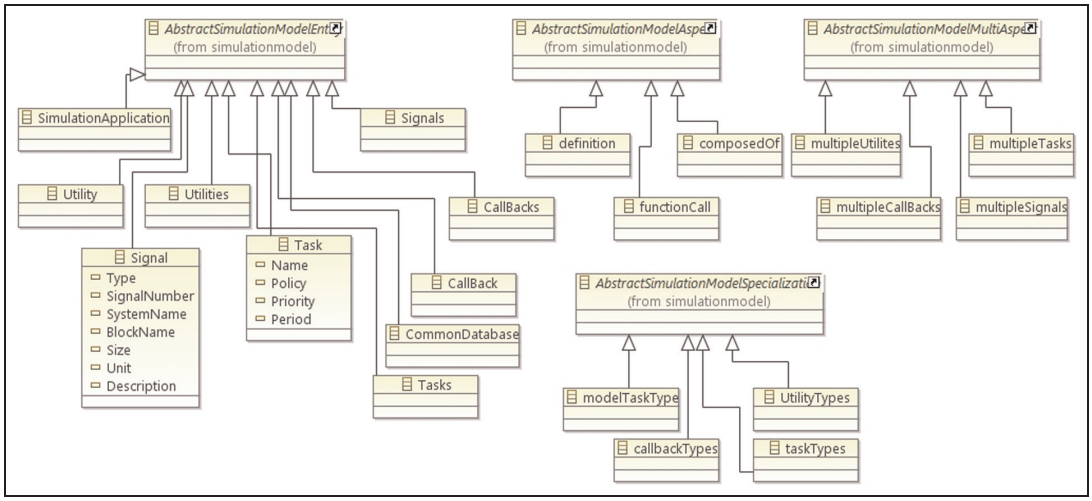


Figure 14. 2Simulate Inheritance class diagram.

Table 1. Task list.

Task type	Name	Policy	Priority	Period
UDPTask	CLS	Round-Robin	60	8 ms
UDPTask	GHS	Round-Robin	60	4 ms
UDPTask	DISP	Round-Robin	51	10 ms
VisualizationTask	VISUAL	Round-Robin	60	4 ms
CANTask	COCKPIT	Round-Robin	57	10 ms
UDPTask	SOUND	Round-Robin	59	100 ms

The introduced metamodel explicitly specifies the real-time distributed simulation domain model of DLR FT by extending SKDM with 2Simulate package. The metamodel elements extend the abstract classes specified in the simulation model package. The rest of this sample case will present how a 2Simulate metamodel can be employed to recover information from an existing simulation software.

Legacy FHS ground-based simulator interface computer software will be analyzed for knowledge recovery. Klaes introduced this interface computer as simulation software to collect relevant simulation data, pilot commands and interpret and process them to emulate subsystem IOs, sensor signals, and status data.[3] Following a manual procedure, existing code has been analyzed to identify the constructs defined in the metamodel. Table 1 shows an extract from this analysis.

While it is also possible to present the recovered information using a graphical model representation, such as a UML profile that would be developed based on the 2Simulate metamodel, a simple table representation is used for the sake of simplicity and to keep the focus on the content and the structure of the data recovered, rather than its representation.

The interface computer corresponds to the simulation application in the 2Simulate metamodel. It has six tasks and about two thousand signals to define the CommonDatabase and a MonitorUtility. Table 1 illustrates the tasks that are recovered from the interface computer. Control Loading System (CLS) is a UDPTask that enables a UDP connection with a control loading system. It is scheduled with Round-Robin, its priority is 60, and its period is 8 ms. The interface computer runs on a QNX real-time operating system. QNX provides each thread with a scheduling priority range from 1 (lowest) to 63 (highest) and runs a thread scheduled with a Round-Robin policy until it voluntarily relinquishes control or is preempted by a higher priority thread and finally consumes its time slice.[43] Likewise, DISP task is another UDP task that enables connection to cockpit displays. The period of DISP task is 10 ms, scheduled again by Round-Robin with a priority 51. The code excerpt from the FHS ground-based simulator interface computer software for the DISP

```
//################################
// Display task
//################################
TSimUdpTask *disp = new TSimUdpTask( hic, "DISP", TASK_SCHED_RR, 51, 10*HIC_SPEEDUPINV*iMSECtoNSEC, true, true, true );
disp->setDesc( "data output to display" );
disp->setHost( "192.168.0.58", 0, 25781 ); // PC20
disp->setSndBuffer( sizeof(COM_OUT_RAW_DISP), (void *)&com->o.r.disp, true );
disp->setPreProcCB( (void(*)(TSim *,TSimRtTask *))&dpDispPre );
```

Figure 15. Code excerpt for DISP task.

Table 2. Task callback functions.

Task	Pre Init	Post Init	Pre Process	Intermediate Process	Post Process
CLS	—	—	dpClsPre	—	dpClsPost
GHS	—	—	dpGHSEcPre	—	dpGHSEcPost
DISP	—	—	dpDispPre	—	—
VISUAL	—	—	dpVisPre	—	—
COCKPIT	—	CanIoPostInit	CanIoUpdateInputs	—	CanIoUpdateOutputs
SOUND	—	—	dpSoundIoUpdateOutputs	—	—

Table 3. Signal list excerpt.

Number	System	Block	Signal	Type	Size	Unit	Description
15000	cls	sp	PosPit	Float	1	deg	Controller position in pitch axis
15001	cls	sp	ForcePit	Float	1	N	Controller force in pitch axis
15002	cls	sp	PosRoll	Float	1	deg	Controller position in yaw axis
15003	cls	sp	ForceRoll	Float	1	N	Controller force in roll axis

task is illustrated in Figure 15. The rest of the tasks can be read similarly. To introduce the tasks, GHS enables a UDP connection to a generic helicopter simulation, which is a helicopter flight dynamics model that runs on a standalone platform, VISUAL task enables out-of-the-window visualization, COCKPIT task enables a connection to cockpit devices and, lastly, SOUND task enables a connection to standalone sound simulation.

Table 2 shows the list of callback functions related to the tasks. As defined in the metamodel, five types of callback functions (PreInitCallBack, PostInitCallBack, PreProcessCallBack, IntermediateProcessCallBack, and PostProcessCallBack) are searched in the interface computer code base. As examples from the list, DISP task only implements a preprocess callback, namely dpDispPre (Figure 15); on the other hand, COCKPIT task implements a post-initialization callback as well as pre-process and post-process callbacks, namely CanIoPostInit, CanIo UpdateInputs, and CanIoUpdateOutputs.

Table 3 gives four sample signals that come from the cyclic control of the FHS safety pilot who sits on the right-hand side.[3] These signals are controller position and force in pitch and roll axis. As a sample signal, the controller position signal for the pitch axis is named as PosPit, its number is 15,000, and it is originated from the CLS and SP (safety pilot) block. The signal has a length of one (1) with a data type float. Its unit is degrees.

5. Conclusion

This paper presents the simulation knowledge discovery metamodel, an extension to KDM to support knowledge discovery from existing simulations for enabling architecture-driven modernization of simulations. Simulation modernization is defined as the evolutionary reengineering of existing simulation software. Unlike the related work on legacy simulation assets that target reusing them "as is" by employing various interfacing technologies, this effort is targeted at supporting simulation modernization. The contribution of this effort can be considered as providing a metamodeling approach for defining meta constructs for knowledge discovery in simulation assets.

Although ADM provides a model-based methodology and KDM provides a metamodel for software modernization, they lack in providing adequate means to recover

knowledge, particularly about simulation from existing simulation software. On the other hand, the composition of the knowledge about the simulation varies to a great extent, depending on the utilized simulation methodology and approach.

This paper proposes a unique extension to OMG's KDM based on the SES that stems from the theory of modeling and simulation. In this endeavor, KDM is augmented with a simulation model package based on SES. Thus, the user is equipped with super-classes to develop metamodels for the particular methodology and approach that is employed for the existing simulation software.

The extended KDM, namely SKDM, is utilized in three diverse representative sample cases; one for a particular methodology (discrete event simulation), one for a particular simulation modeling language (Modelica) and one for a particular organization (DLR FT). These examples provide valuable evidence that shows that, with a simulation model package, SKDM equips its users with adequate meta constructs to develop metamodels for their diverse simulation methodologies and approaches. Then, using these metamodels, it is possible to recover simulation knowledge from existing simulation software.

This paper introduces the simulation model package and shows how it is used to develop a DeMO package, Modelica package, and 2Simulate package. It explains these packages as other packages that are explained in KDM.

Future work includes establishing the computational means for these declarative specifications in order to automate processes in the horseshoe model. It is planned to develop a parser for a 2Simulate package to extract knowledge from DLR FT's existing simulation software. As the next step, it is intended to develop a 2Simulate UML profile and transformation scripts to construct a UML-based representation of this knowledge in order to enhance human readability. Enhanced understanding of existing simulation software will enable better maintenance and feature enhancement. As the last step, it is envisioned to establish forward engineering tools to construct a fully architecture-driven modernization pipeline.

Funding

This research received no specific grant from any funding agency in the public, commercial, or not-for-profit sectors.

References

1. Saager P. Real-time hardware-in-the-loop simulation for 'ATTAS' and 'ATTHeS' advanced technology flight test vehicles. In: *50th symposium on AGARD guidance and control panel*, Izmir, Turkey, 22–25 May 1990.
2. Klaes S. ATTAS ground based system simulator – an update. In: *AIAA Modeling and simulation technologies conference and exhibit*, Denver, CO, 14–17 Aug 2000.
3. Kleas S. ATTAS & ACT/FHS system simulation for preflight software and hardware testing. In: *AIAA modeling and simulation technologies conference and exhibit*, Monterey, CA, 05–08 Aug 2002.
4. Duda H, Gerlach T, Advani S, et al. Design of the DLR AVES research flight simulator. In: *AIAA modeling and simulation technologies (MS) conference*, Boston, MA, 19–22 Aug 2013.
5. Vansteenkiste G and Kerckhoffs E. Architectures for simulation environment. In: *Scientific Computing on Super Computers*, New York: Plenum Press, 1989, pp.47-69.
6. Zwaanenburg K. Six degree-of-freedom missile simulation using the ADI AD 100 digital computer and ADSIM simulation language. In: *Proceedings of the 3rd annual conference on aerospace computational control*, Pasadena, CA, 15 Dec 1989.
7. Gerlach T, Durak U and Gotschlich J. Model Integration workflow for keeping models up to date in a research simulator. in *Simultech14*, Vienna, Austria, 28–30 Aug 2014.
8. Trcka Radosevic M, Hensen J and Wijsman A. Distributed building performance simulation: a novel approach to overcome legacy code limitations. *HVAC&R Res* 2006; 12: 621-640.
9. Pullen J and White E. Adapting legacy computational software for XMSF. In: *Fall 2003 simulation interoperability workshop*, Orlando, FL, 14–19 Sep 2003.
10. Sonntag M, Hotta S, Karastoyanova D, et al. Using services and service compositions to enable the distributed execution of legacy simulation applications. in *ServiceWave 2011*, Poznan, Poland, 26–28 Oct 2011.
11. Lacy L. *Interaction models using the web ontology language – OWL*. Orlando, FL: University of Central Florida, 2006.
12. Lehman M. On understanding laws, evolution, and conservation in the large-program life cycle. *J Syst Software* 1980; 1: 213–221.
13. Lehman M. Programs, life cycles, and laws of software evolution. *Proc IEEE* 1980; 9: 1060-1076.
14. Visaggio G. Ageing of a data-intensive legacy system: symptoms and remedies. *J Software Maintenence Evol Res Pract* 2001; 13: 281-308.
15. Parnas D. Software aging. In: *Proceedings of the 16th international conference on software engineering*, Sorrento, Italy, 16–21 May 1994.
16. Lehman M, Perry D and Ramil J. Implications of evolution metrics on software maintenance. In: *International conference on software maintenance*, Bethesda, MD, 16–20 Nov 1998.
17. Chikofsky E and Cross J. Reverse engineering and design recovery: a taxonomy. *IEEE Software* 1990; 7: 13-17.
18. Sneed H. Estimating the costs of a reengineering project. In: *12th working conference on reverse engineering*, Pittsburgh, PA, 7–11 Nov 2005.
19. Object Management Group. Architecture-driven modernization, http://adm.omg.org/legacy/ADM_whitepaper.pdf. (2003, accessed 5 May 2014).
20. Object Management Group. *Knowledge discovery metamodel (KDM) version 1.3*. Needham, MA: Object Management Group, 2011.
21. Silver G, Miller J, Hybinette M, et al. DeMO: an ontology for discrete-event modeling and simulation. *Simulation* 2011; 87: 747-773.

22. Kim T, Lee C, Christensen E, et al. System entity structuring and model base management. *IEEE Trans Syst Man Cybern* 1990; 20: 1013-1024.
23. Tolk A. Avoiding another green elephant – a proposal for the next generation HLA based on the model driven architecture. in *Fall simulation interoperability workshop*, Orlando, FL, 8–13 Sept 2002.
24. Durak U, Oguztuzun H and Ider K. Ontology based domain engineering for trajectory simulation reuse. *Int J Software Eng Knowledge Eng* 2009; 9: 1109-1129.
25. Topçu O, Adak M and Oğuztüzün H. A metamodel for federation architectures. *Trans Model Comput Simul* 2008; 18: 10:1-10:29.
26. Cetinkaya D, Verbraeck A and Seck M. MDD4MS: a model driven development framework for modeling and simulation. In: *Summer simulation conference*, The Hague, Netherlands, 27–30 Jun 2011.
27. Mattsson S, Elmqvist E and Otter M. Physical systems modeling with Modelica. *Control Eng Prac* 1998; 6: 501-510.
28. Gotschlich J, Gerlach T and Durak U. ''2Simulate: a distributed real-time simulation framework. In: *ASIM STS/GMMS workshop 2014*, Reutlingen, Germany, 20–21 Feb 2014.
29. Object Management Group. *Why do we need standards for the modernization of existing systems?* Needham, MA: Object Management Group, 2003.
30. Bergey J, Smith D, Weiderman N, et al. *Option analysis for reengineering (OAR): issues and conceptual approach*. Pittsburg, PA: Carnegie Mellon Software Engineering Institute, 1999.
31. Khusidman V. ADM transformation white paper V.1., www.omg.org/adm/ADMTransformartionv4.pdf (2008, accessed 15 May 2014).
32. Perez-Castillo R, de Guzman I and Piattini M. Architecture-driven modernization. In: *Modern software engineering concepts and practices: advanced concepts*. Hershey, PA: International Science Reference, 2011, pp.75-103.
33. Moyer B. Software archeology: modernizing old systems. *Embedded Technol J* 2009; 17: 1-4.
34. Durak U, Oguztuzun H and Ider S. An ontology for trajectory simulation. In: *Proceedings of the 2006 winter simulation conference*, Montarey, CA, 03–06 Dec 2006.
35. Ören T and Zeigler B. System theoretic foundations of modeling and simulation: a historic perspective and the legacy of A Wayne Wymore. *Simulation* 2012; 88: 1033-1046.
36. Rozenblit J and Zeigler B. Representing and construction of system specifications using the system entity structure concepts. In: *Proceedings of the 1993 winter simulation conference*, Los Angeles, CA, 12–15 Dec 1993.
37. Zeigler B and Hammonds P. *Modeling and simulation-based data engineering: introducing pragmatics into ontologies for net-centric information exchange*. London: Academic Press, 2007.
38. Lee H and Zeigler B. System entity structure ontological data fusion process integrated with C2 systems. *J Defense Model Simul Appl Methodol Technol* 2010; 4: 206-225.
39. Zeigler B. *Object-oriented simulation with hierarchical, modular models: inteligent agents and endomorphic systems*. San Diego, CA: Academic Press Professional, 1990.
40. Zeigler B. *Multifaceted modeling and discrete event simulation*. Orlando, FL: Academic Press, 1984.
41. Modelica Association. *Modelica – a unified object-oriented language for systems modeling language specification, version 3.3*. Linköping, Sweden: Modelica Association, 2012.
42. Object Management Group. *SysML-Modelica transformation version 1.0*. Needham, MA: Object Management Group, 2012.
43. QNX Software Systems GmbH Co. KG. *QNX Neutrino operating system: programmer's guide for QNX Neutrino 6.4*. Ontario, Canada: QNX Software Systems International Cooporation, 2008.

Author biography

Umut Durak is currently participating in the Flight Dynamics and Simulation Department of the German Aerospace Center (DLR) Institute of Flight Systems as a research scientist. There, he conducts simulation engineering research on application of model-based methodologies in flight simulators and flight simulator validation and verification. In addition, he offers graduate-level courses on simulation at TU Clausthal, Department of Informatics as an adjunct lecturer. He received his BS, MS and PhD degrees in mechanical engineering from Middle East Technical University (METU). In the last 15 years, he has contributed various research and development projects and has published more than 30 papers in national and international conferences, workshops and journals. He is a member of the Society for Modeling and Simulation International (SCS), Arbeitsgemeinschaft Simulation (ASIM) and the American Institute of Aeronautics and Astronautics (AIAA), and he is a part of AIAA Modeling and Simulation Technical Committee and editorial teams of the *Journal of Defense Modeling & Simulation* and the *International Journal of Modeling, Simulation, and Scientific Computing*.

B.11 Pragmatic Model Transformations for Refactoring in Scilab/Xcos

Umut Durak. 2016. Pragmatic model transformations for refactoring in Scilab/Xcos. *International Journal of Modeling, Simulation, and Scientific Computing* 7,1, Article 1541004, 23 pages.

PRAGMATIC MODEL TRANSFORMATIONS FOR REFACTORING IN SCILAB/XCOS

UMUT DURAK
German Aerospace Center (DLR), Institute of Flight Systems, Lilienthalplatz 7
Braunschweig, 38108, Germany
umut.durak@dlr.de

Model-Based Development has become an industry wide standard paradigm. As an open source alternative, Scilab/Xcos is being widely employed as a hybrid dynamic systems modeling tool. With the increasing efficiency in implementation using graphical model development and code generation, the modeling and simulation community is struggling with assuring quality as well as maintainability and extendibility.

Refactoring is defined as an evolutionary modernization activity where, most of the time, the structure of the artifact is changed to alter its quality characteristics, while keeping its behavior unchanged. It has been widely established as a technique for textual programming languages to improve the code structure and quality. While refactoring is also regarded as one of the key practices of model engineering, the methodologies and approached for model refactoring are still under development.

Architecture-Driven Modernization has been introduced by the software engineering community as a model-based approach to software modernization, in which the implicit information that lies in software artifacts is extracted to models and model transformations are applied for modernization tasks. Regarding refactoring as a low level modernization task, the practices from Architecture-Driven Modernization are adaptable. Accordingly, this paper proposes a model based approach for model refactoring in order to to come up with more efficient and effective model refactoring methodology that is accessible and extendable by modelers.

Like other graphical modeling tools, Scilab/Xcos also possesses a formalized model specification conforming to its implicit metamodel. Rather than proposing another metamodel for knowledge extraction, this pragmatic approach proposes to conduct in place model-to-model transformations for refactoring employing the Scilab/Xcos model specification. To construct a structured model-based approach, the implicit Scilab/Xcos metamodel is explicitly presented utilizing ECORE as a meta-metamodel. Then a practical model transformation approach is established based on Scilab scripting. A Scilab toolset is provided to the modeler for in place model-to-model transformations. Using a sample case study, it is demonstrated that proposed model transformation functions in Scilab provide a valuable refactoring tool.

Keywords: Model Refactoring; Scilab/Xcos; Model Engineering; Model Transformations.

1. Introduction

Model-Based Development (MBD) has been described as a development paradigm characterized by seamless utilization of graphical models for specification, design and implementation [1]. While the common practice is employed over commercial modeling and simulation tools, such as MATLAB/Simulink [2], Scilab/Xcos [3] is

enlarging its user base as an open source alternative. It is promoted as a tool for modeling and simulation of hybrid dynamic systems with continuous and discrete components.

MBD proposes a specification and design in a graphical modeling environment via constructing executable models. On the other hand, implementation is enabled via code generation facilities of the employed toolset. With MBD, models become the main artifacts. The quality characteristics of the product become tightly coupled with the quality characteristics of the models. However, while MBD provides an effective development methodology that leads to construction of complex system models and controller algorithms in short development cycles, structuring and maintaining the model for better quality become labor intensive and time consuming [4].

Model engineering addresses the credibility of the whole model life cycle with the minimum cost with a systematic, standardized, quantifiable methodology which consists theories, processes, technologies, standard and tools [5]. Zeigler and Zhang introduce model refactoring or reconstruction, as one of the key model engineering technology. They describe it as adjusting the internal structure without changing the external functions of the models with an aim to optimize the model performance, understandability, maintainability and adaptability [6].

The German Aerospace Center (DLR) Institute of Flight Systems employs MBD for both flying platforms [7] and ground based simulators [8]. Model maintenance and structuring is a constant activity of model engineering in a research environment. Whilst new features are added to flight models for particular research objectives, the quality of the models are sought to be preserved by an extensive maintenance effort. Scilab/Xcos, on the other hand, provides no particular approach other then ad hoc, manual manipulation for refactoring which is inherently labor intensive and error prone. The motivation of this endeavor is to develop an efficient and effective model refactoring approach that is accessible, maintainable and adoptable by modelers.

Refactoring has been used in classical software development as an evolutionary modernization technique in order to incrementally alter the structure of the artifact to achieve a better quality, while keeping its behavior unchanged. In their reverse engineering taxonomy, Chikofsky and Cross [9] classified refactoring as one of the appearance of restructuring which is essentially a transformation from one form to another at the same abstraction level while maintaining functionality and semantics. Fowler described refactoring as a process of cleaning up the code to improve its design after it has been written [10].With refactoring, code is tidied up in order to keep its shape.

Like other modernization efforts, refactoring also adheres to the horseshoe model that was introduced by the Software Engineering Institute in the late 90s. This model defines three processes: the first, the analysis of an existing system, goes up the left leg; the second, the logical transformation goes across the top of the horseshoe; and the last, the development of the new one, goes down the right leg

of the horseshoe [11].

While manual approaches are always applicable to modernization, model based approaches bring increased efficiency and effectiveness and enhanced maintainability. Architecture-Driven Modernization (ADM) has been introduced by the Object Management Group (OMG) as a model-based approach for comprehending and transforming existing software assets [12]. The horseshoe metaphor is adopted by introducing three levels that represent the level of abstraction to be reached during modernization: technical modernization, application/data modernization and business modernization [13]. While technical modernization has been defined as the most common modernization effort that targets mostly language or platform changes, application/data modernization is introduced as an effort that targets a change in system design. And, lastly, business modernization is presented as one that attacks the business rules and processes that are governed by the software. OMG ADM Task Force claimed that any modernization effort, no matter at what abstraction level, follows the horseshoe model, but requires different tools. We can locate refactoring at the lowest abstraction level and regard it as a technical modernization.

Following the OMG ADM Task Force's claim, model refactoring requires particular tools. A metamodel is needed as the basis of the formal specification of the model [4]. Reverse engineering, transformation and forward engineering tools are required to specify the parts to be restructured, to extract them from the model, restructure them using transformations and reflect the changes to the model again.

While there is well-established research on code refactoring [10, 14], there are relatively few efforts on model refactoring for MBD [4,15–20]. While they present a particular interpretation of the horseshoe model, unfortunately, all of these efforts target at MATLAB/Simulink. Even though there is a wide range of simulation modeling environments that support particular methodologies and languages, like CosMos [21], MS4 Me [22] and PowerDevs [23] for DEVS models, or Open Modelica [24] and JModelica [25] form Modelica models, refactoring has not been addressed for any of these tools and environments.

Furthermore, the accessibility of the proposed approach and the maintainability and adaptability of the proposed tool set are regarded as the key factors for the successful implementation and will be the basis for discussions about the related work.

As with other graphical modeling tools, Scilab/Xcos also possesses a formalized model specification conforming to its implicit metamodel. Rather than proposing another metamodel for knowledge extraction, this pragmatic approach proposes to utilize the Scilab/Xcos metamodel and conduct in place model-to-model transformations for refactoring. The Scilab/Xcos metamodel is explicitly presented at first. The physical representation of the metamodel is regarded as the Scilab model structure scs_m [26]. The paper proposes a practical approach in order to provide modelers with Scilab script level Application Programming Interface (API) to en-

hance accessibility, maintainability and adaptability. The proposed API possesses composite model transformation functions that execute overall model transformation, as well as the atomic model transformation functions for *find*, *add*, *delete* and *replace* type basic operations on the blocks and the links. Thus, while the modelers are equipped with a ready to use refactoring functions, they are also provided with building blocks for developing their tailored applications. As a pragmatic decision, all functions basically manipulate the Scilab model structure scs_m, thus enable a native and seamless reverse and forward engineering. The introduced methodology is exercised with a prototype implementation in a case study in order to present evidences of its usability.

The paper is organized as follows. Section 2 introduces the model refactoring approach by discussing related previous work. In section 3 the Scilab/Xcos metamodel is presented. Details about the specification and model transformation will be introduced in Section 4. Section 5 will present a sample case study. Lastly a conclusion is presented in section 6.

2. Model Refactoring

2.1. *Refactoring: An Overview*

Mens and Tourwe [14] describe the refactoring process with six activities: identifying the refactoring point, determining the refactoring method, guaranteeing the preserved behavior, applying the refactoring, assessing the effect of refactoring on the quality and maintaining the consistency of refactored code and the documentation.

Identification of the program parts that require refactoring is described as detection of bad smells [10]. Clone detection is one of the examples for detection of bad smells. Since the code duplication reduces maintainability, it has long been regarded as a bad smell and various techniques have been developed to detect duplicated code [27]. There are also some efforts in MBD community for developing model clone detection tools and techniques for MATLAB/Simulink [28–31]. Further model checking is a well applied approach in MBD in order to detect bad smells with identifying the violations of modeling guidelines [20].

The definition of refactoring makes it clear that the refactoring activity shall not change the behavior of the software. While the complete specification of behavior is not easy, testing provides evidences to guarantee that the behavior is preserved. Automated testing approaches like model-based testing can yield an effective practice to define and execute test cases. Model-based testing tools and techniques in MBD community [32,33] can be applied to check the preserved behavior after model refactoring.

This effort targets particularly the activity of applying refactoring. Ad hoc and manual manipulation of models is always possible for refactoring purposes, but it is labor intensive and error prone. Additionally, manual refactoring tasks are hardly

repeatable. Model transformations were proposed as a structured way to define refactoring tasks and execute them over the models of interest [34, 35]. While executing predefined model transformations provide efficiency via automation, defining model transfomations provided an effective way of refactoring. This idea is applied for refactoring MATLAB/Simulink models [4, 15–20]. This paper presents an effort in this track that utilizes model transformations for refactoring Scilab/Xcos models.

In MBD, model checking and clone detection tools and techniques can further be employed to check the effect of refactoring on model quality. Since MBD proposes models as main artifacts, consistency between the model and the code is maintained inherently by code generation process.

Fowler catalogs 72 different code refactorings. Some examples are renaming a method, adding a parameter or replacing a constructor with factory method. Sunye et al. introduce five types of refactoring operation in UML class diagrams, namely addition, removal, move, generalization and specialization [36]. They apply on the features of like attributes, methods and associations. Likewise, typical model refactorings in MBD tools can be listed as removing, replacing and adding single or a pattern of model features like blocks or links. Deleting unconnected blocks, block inputs and block outputs for maintenance purposes or deleting sink blocks for simulation control like END or HALT blocks before code generation can be given as example delete tasks. Adding scopes for instrumentation purposes after or before all integrator blocks can be pronounced as an addition example. Finally replacing a pattern with a user defined library block can be provided as an example of a typical replace refactoring.

2.2. *Related Work*

In 2006, Giese, Meyer and Wagner proposed to utilize the model transformation capabilities of Fujaba [37] for Simulink model refactoring. They aimed at refactoring or repairing the model to comply with generally accepted MATLAB/Simulink modelling guidelines [20]. Giese et al. claim rule-based model checking and repairing methodologies, like the Simulink Model Advisor [38], are disadvantageous, because they require extensive programming and maintenance effort. Hence, they propose a higher abstraction description of guidelines as a better approach. Fujaba is an open source model-based software engineering tool. The graph transformation capabilities of Fujaba are employed for formally specifying graphical rules, the so-called pattern rules for guideline violations, which refer to the MATLAB/Simulink metamodel. Fujaba is started within MATLAB/Simulink. The inference algorithm applies the rules to the MATLAB/Simulink model. Fujaba requires a conformance to its own metamodel format and rules to navigate the model elements, to modify them as well as to create new ones. Hereby, the authors constructed a rudimentary Fujaba compliant metamodel for MATLAB/Simulink. Metamodel implementation was carried out using Fujaba's code generation facilities. To link the MATLAB/Simulink models and the Fujaba-generated code, an adapter was developed.

This study solved the interfacing problems between Fujaba Tool Suite, which enables definitions of guidelines in higher abstraction level. But, as also reported by the authors, the proposed approach requires the rule developer to be familiar with pattern language and the MATLAB/Simulink metamodel. On the other hand, complex and large pattern rules developed using Fujaba turned out to be not readable and maintainable. Therefore, Giese et al. attempted to develop an approach to model guideline violations using concrete MATLAB/Simulink syntax.

As a follow up study, in the Model Advisor Transformation Extension (MATE) approach, Stürmer et al. [16–18] proposed a graph-based description of guideline violations in the model and introduced the transformations to repair them. MATE is based on the Mathworks Automotive Advisory Board's Control Algorithm Modeling Guidelines Using MATLAB, Simulink, and Stateflow [39]. MATE provides automatic repair functions, interactive repair functions, design pattern instantiations and model beautifying operations for conforming these guidelines. They follow the approach from [20]. The guideline violations are described as patterns utilizing a metamodel for MATLAB, Simulink and Stateflow diagrams. Graph transformations are used as repair mechanisms. Since these transformations exceed the boundaries of Matlab during execution, it may harm the accessibility of the methodology and its adaptability and maintainability by the typical Matlab user.

The following two studies were implicitly targeted at Matlab users' capability set. Tran et al. [4] proposed a metamodel for Simulink to conduct model-based refactoring. Then they defined the basic transformations and modification steps that run over the instances of their Simulink metamodel as the building blocks of complex refactoring operations. These basic functions are implemented as MATLAB functions and are made available to the user as a high level API. *Add block*, *add input port*, *add output port*, *replace block* and *delete block* are some of these functions. The basic preconditions for selecting blocks in this study are the positions of the blocks. While the API in this study provides the all necessary building blocks to script a tailored refactoring, a generic model rewriting functionality for model patterns is not introduced.

Denil et al. [15] introduced a rule-based model transformation for MATLAB/Simulink models that is specified and executed in Simulink. They also employed a third party transformation library T-Core, but provided generic model rewriting functions as blocks in MATLAB/Simulink and enhanced the accessibility and adaptability for the user. These generic functions correspond to the transformation rules, namely *atomic rule*, *for all rule* and *star rule*. They accept precondition and post condition blocks, which can be regarded as Left Hand Side (LHS) and Right Hand Side (RHS) of the transformation as arguments. The atomic rule finds a single instance of the precondition and replaces it with the post-condition. While *for all rule* finds the all instances of the precondition and executes the transformation, *star rule* makes its search recursively until no precondition is found. Although the building blocks of these generic functions are not available, generic functions enable their user to design their own transformation to some extent and provide them

with a basic understanding of the transformation process applied on the model.

The following section introduces how the current approaches available in the literature, basically for MATLAB/Simulink, are enhanced with the proposed model refactoring approach for Scilab/Xcos.

2.3. *Proposed Model Refactoring Approach*

Refactoring is a labor intensive and repetitive task. Thus, with a model based approach, we would like to promote model developers to develop modification scripts following a well-established methodology. We would like to provide model developers with the capability to transform their Scilab/Xcos models for refactoring purposes using the proposed API. There are various model transformation approaches. Please refer to Czarnecki et al. [40] for their classification.

First, the metamodel of Scilab/Xcos is presented to make its abstract syntax explicitly available as the baseline of the model transformation. Hereby, the scs_m structure [26] [41] that captures the overall data for Xcos models is represented as a metamodel. Model transformations are defined by a precondition pattern, which can be described as the LHS and the outcome pattern, which can be introduced as the RHS. This study follows an in-place transformation approach. Thereby, the transformation operation comprises matching the LHS pattern in the model being transformed and replacing it with the RHS pattern in place [40]. Kühne and co-workers propose that pattern specification metamodels can be obtained by subjecting the original language metamodel to relaxation, augmentation and modification [42]. Accordingly, in this study, the LHS pattern specification approach is derived from the Xcos metamodel employing relaxation and augmentation in the abstract level and the scs_m structure is adapted in a concrete level. Search patterns are proposed to be developed using the scs_m structure. As the augmentation, in order to define the constraints, regular expressions are proposed as the values of attributes in LHS pattern structure. As a relaxation, since not all the fields of Xcos metamodel are suitable for constraint definition, the Xcos metamodel is simplified for refactoring purposes. Further, in the simplification, all the data types of the parameter values are specified as strings in order to enable the application of regular expressions. The RHS pattern, which is the replace pattern, is proposed to be specified using the same structure with the model conforming to Xcos Metamodel. While it is not fully alright to use the original language definition since replace pattern usually fails to fulfill the constraints concerned with the completeness of the model [42], pragmatically in this study, RHS patterns are regarded as model fragments and completeness is not required from them.

Eclipse Modeling Framework (EMF) [43] is utilized as the metamodeling tool set. Eclipse can be introduced as an open source platform for the implementation of integrated development environments (IDEs). EMF, on the other hand, provides a basic framework for modeling in context of supporting model-based development technologies in Eclipse. The model that is used to define models in EMF is called

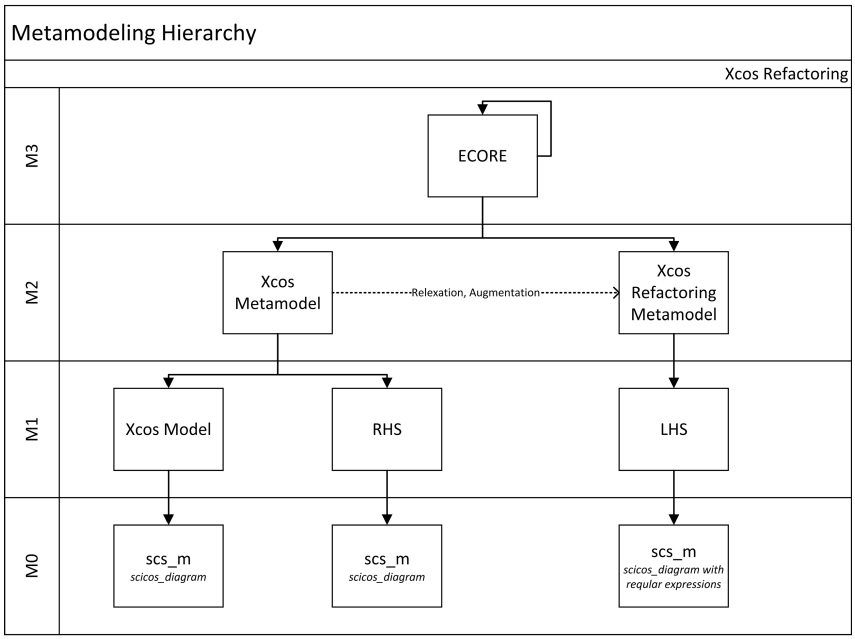


Fig. 1. Metamodeling Hierarchy

ECORE. It is the meta-metamodel of the EMF environment that is introduced to develop metamodels [44]. Here in this study, ECORE is employed as the meta-metamodel. As presented in Figure 1, ECORE is at the Meta-Metamdeling (M3) level of the metamodeling hierarchy. Two metamodels, Xcos Metamodel and Xcos Refactoring Metamodel, are described using ECORE in Metamodel (M2) level. The reader will find the Xcos models and RHS patterns that are specified using Xcos Metamodel, and the LHS pattern that is defined using Xcos Refactoring Metamodel in Model (M1) level. In the physical level (M0), scs_m data structure that represents scicos_diagram will be used for both Xcos and RHS pattern and relaxed scs_m with regular expressions as attribute values for LHS patterns.

Not perfectly aligned with Denil et al. [15], who propose overall transformation functions as atomic transformation, all transformation and recursive transformation, this study finds it better to maintain the *find*, *add*, *delete* and *replace* feature set for understandability. Hereby, the overall model transformation functions of the API are proposed as *finding*, *adding*, *deleting* and *replacing* a sub-diagram in a Scilab diagram. These composite functions are implemented using the atomic model transformation functions. Referring to Tran et al. [4], these atomic functions are proposed for *finding*, *adding*, *deleting* and *replacing* a single block and, a single link. Further functions are proposed to find the connected blocks to a block and the links between any two blocks. API is proposed as the collection of atomic and

composite functions.

The approach that is presented in this paper differs from the approach present Denil et al. [15] as it provides a scripting API that manipulates the scs_m structure for in place model transformation within Scilab. As all the model manipulation is carried out in Scilab environment, all the steps of the transformation are more transparent and accessible. Thus, they are easily modifiable and extendable. On the other hand, it differs from Tran et al. [4], since the scripting API is constructed based on model transformation concepts. Thus, it intends to bring model transformation practices within the capability set of modelers.

The metamodels are presented in Section 3, Metamodeling. The methodology for specifying the search and replace patterns, the functions that are proposed in the API and details of model transformations will be introduced in Section 4, Specification and Transformation.

3. Metamodeling

In the metamodeling section both the Xcos Metamodel and the relaxed version of it, namely, Xcos Refactoring Metamodel will be introduced.

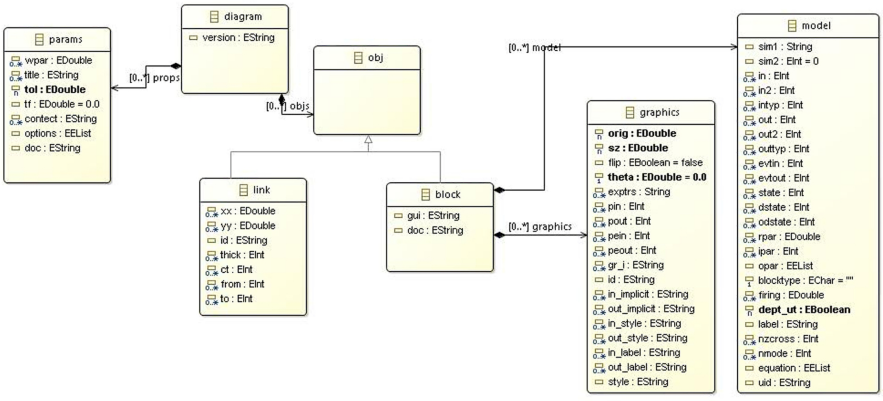


Fig. 2. Xcos Metamodel

The scs_m basically corresponds to the Xcos diagram. As presented in Figure 2, any Xcos diagram is composed of diagram properties, a list of objects in the Xcos diagrams and a version string. Parameters are defined using the *param* structure. Some of the important attributes of the *param* are as follows. The *title* is composed of two strings; the first is for the diagram title and a default name of save filename, and the second is the path of the directory where the model is saved. The *tol* keeps a vector of real values with the size of 7 referring to the parameters of the simulation. While the first value is the absolute tolerance for the solver, the second value is for

the relative tolerance. The list includes the solver type, time scaling and maximum step size. The *tf* field represents the final time for the simulation.

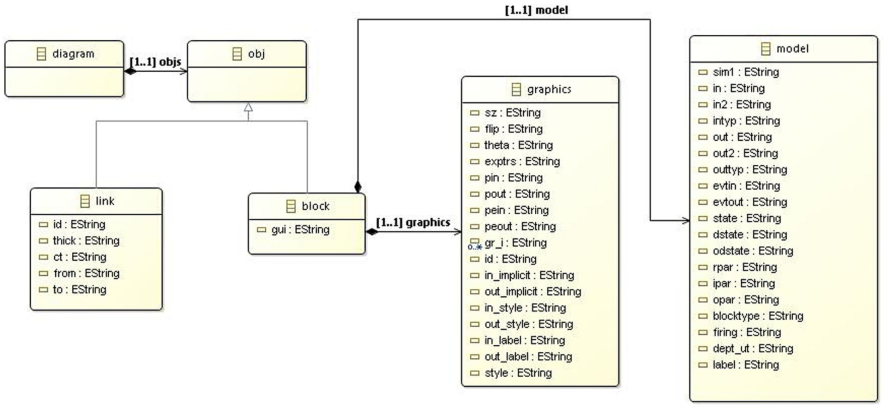


Fig. 3. Xcos Refactoring Metamodel

There are two types of objects in scs_m, *link* and *block*. *Link* defines a link structure between blocks. Its attributes include *xx* and *yy* for the coordinates of the link path, *id* for the link identification, *thick* for its thickness, *ct* for its color and, *from* and *to* in order to describe block number, port number and the port type at the origin and the destination. *Block*, on the other hand defines a block structure which is composed of a *graphics* object for the graphical features of the block and *model* object for the compilation features of the block. Some of the important attributes of the *graphics* object are *orig*, which represents the origin of the block, *size*, which represents its size; and *theta*, which represents the angle of the block. The important compilation features that are captured in the *model* object are inputs and outputs, which are captured by *in*, *in2*, *intyp*, *out*, *out2*, *outtyp*, *evtin* and *evtout* and the *state* of the block, including the continuous time state and the discrete time state. Parameters of the block are also captured in the model object attributes *rpar*, *ipar* and *opar*.

While the Xcos Metamodel captures all data elements of scs_m as well as their accurate data types, Xcos Refactoring Metamodel (Figure 3) is relaxed via picking only a subset of objects and object properties that will be used to define constraints in the LHS. As an example, *id*, *thick*, *ct*, *from* and *to* are the attributes that are kept for the link object. Further, as all the constraints will be specified as regular expressions, all data elements are assigned a string data type.

The following section will introduce how these metamodels will be used for specification of the LHS and RHS as well as the transformation process.

4. Specification and Transformation

The specification problem in model transformation addresses the definition of the precondition and the post-condition, namely the LHS and the RHS. The variables, patterns and logic are used to specify LHS and RHS [45]. Variables are defined as the elements from the source and target. Patterns are defined as model fragments with zero or more variables. An abstract or a concrete syntax of the source or target modeling language can be employed to define the patterns. Both textual and graphical syntax can be utilized. Logic, on the other hand, holds the constraints on the model elements. It may be either executable or non-executable and executable ones can be either declarative or imperative.

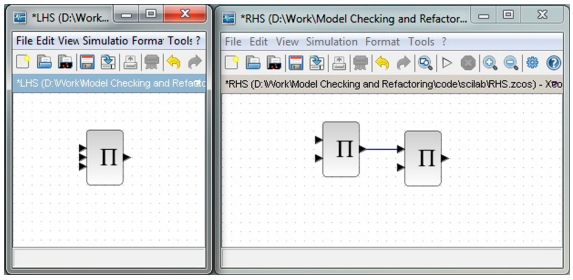


Fig. 4. Example Specification of Left Hand Side and Right Hand Side using Xcos Concrete Syntax

In this study, the variables of the transformation are the objects of the Xcos diagram with their attributes. Then, patterns can be introduced as the composition of these elements. To foster the usability and maintainability, the graphical concrete syntax of Xcos is proposed to define the patterns. An example is presented in Figure 4. Any executable logic specification is recommended that employs regular expressions as the declarative constraints to retrieve elements from the source model. Then the LSH scs_m structure can be altered with scripting to assign a constraint on any variable.

Regular expressions are defined as patterns of characters that are used to find desired pieces of text [46]. They are used in context of another application or a language. Amelunxen et al. introduce them as a tool to check model naming conventions in Simulink [16]. Scilab also supports them with its *regexp* function which the user can call in a Scilab script with an input string and a pattern and receives the starting index of each substring that matches the regular expression as well as the final index and substring itself [47]. As an example, we would like to constrain LHS in Figure 4 in a way that we can filter only PRODUCT blocks with complex inputs from this transformation rule. It is then proposed to describe this constraint as a regular expression in *intyp* attribute of the model in PRODUCT object ins scs_m. Assuming that the values of the attributes are serialized to string values as comma

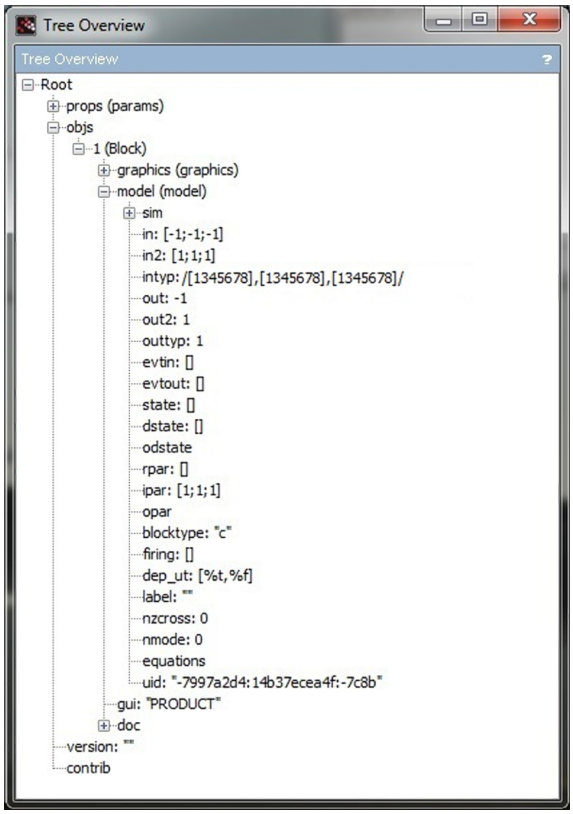


Fig. 5. Representation of Example Regular Expression in Xcos Diagram Browser

separated values, the a regular expression "/[1345678],[1345678],[1345678]/" will specify that we expect three inputs which have a type different than "2", namely a complex matrix. The representation of regular expressions in Xcos Diagram Browser is depicted in Figure 5.

As the proposed approach for the specification of the patterns and the logic is introduced, the next challenge is to conduct the transformation. Here, an application programming interface is proposed for both atomic operations and the overall transformation process.

Atomic functions include finding, adding, deleting and replacing a block and, likewise finding, adding, deleting and replacing a link. Further, getting the list of connected blocks and the list of connecting links between the blocks is also proposed as an atomic function. Table 1 lists all the atomic model transformation functions, with their interfaces, that are specified as input and output arguments.

Here, the *find_block* and *find_link* functions basically conduct the graph search to make the matching for a single block or a link as the LHS pattern. The default

Table 1. Atomic Transformation Functions.

Function name	Input arguments	Output arguments
find_block	scs_m structure for the diagram block structure to be searched list of constraining attributes list of constraints	list of indexes for matching blocks
add_block	scs_m structure for the diagram block structure to be added	scs_m structure for the updated diagram
delete_block	scs_m structure for the diagram index of block to be deleted	scs_m structure for the updated diagram
replace_block	scs_m structure for the diagram index of the block to be replaced new block structure	scs_m structure for the updated diagram
find_link	scs_m structure for the diagram link structure to be searched list of constraining attributes list of constraints	list of indexes for matching links
add_link	scs_m structure for the diagram link structure to be added	scs_m structure for the updated diagram
delete_link	scs_m structure for the diagram index of the links to be deleted	scs_m structure for the updated diagram
replace_link	scs_m structure for the diagram index of the link to be replaced new link structure	scs_m structure for the updated diagram
find_connected_blocks	scs_m structure for the diagram index of the block	list of indexes for matching blocks
find_connecting_links	scs_m structure for the diagram index of the block 1 index of the block 2	list of indexes for matching links

selection criteria for the block is the function name, which is represented in the *gui* attribute and the identification for the link that is represented in the *id* attribute. Not all attributes are used at the same time as constraints. For further selection of the criterion, *list of constraining attributes*, which contains a logic for selection, is added to the *find_xxx* functions. And the constraints are then specified using *list of constraints* argument. These arguments can be regarded as the implementation of Xcos metamodel relaxation and augmentation in the API level. The output of these functions are a list of indexes to the matching entities. Thus, these indexes can be used for delete and replace operations.

The *delete_block* operation is recommended to delete all the connecting links to the block, but, on the other hand, *delete_link* is proposed to delete only the indexed link.

Whilst these atomic model transformation functions will provide the modeler with the building blocks for developing their own algorithms to manipulate or transform their models, they are also used to construct, basically, the composite model transformation functions *find_subdiagram*, *add_subdiagram*, *delete_subdiagram*, *replace_subdiagram* and an overall *find_and_replace*, which are listed in Table 2. The sub-diagram here refers to a collection of blocks and related links. Aligned with the atomic model transformations functions, *find_subdiagram* searches the entire dia-

Table 2. Composite Transformation Functions.

Function name	Input arguments	Output arguments
find_subdiagram	scs_m structure for the diagram scs_m structure to be searched list of indexes for the constraining objects list of constraining attributes list of constraints	list of indexes for matching blocks list of indexes for matching links
add_subdiagram	scs_m structure for the diagram scs_m structure to be added	scs_m structure for the updated diagram
delete_subdiagram	scs_m structure for the diagram list of indexes of the blocks to be deleted	scs_m structure for the updated diagram
replace_subdiagram	scs_m structure for the diagram list of indexes for the blocks to be replaced new scs_m structure	scs_m structure for the updated diagram
find_and_replace	scs_m structure for the diagram scs_m structure to be searched (LHS) list of indexes for the constraining objects (Logic) list of constraining attributes (Logic) list of constraints (Logic) new scs_m structure (RHS)	scs_m structure for the updated diagram

gram for the specified sub-diagram with an scs_m structure. While the default selection constraints apply, any particular logic can be specified using the list to define the object, its attribute and the constraints as regular expression. *find_subdiagram* returns a list of indexes for the first matching objects and links. The indexes can be used to delete and replace the sub-diagram. Furthermore, the *find_and_replace* function is proposed for the canonical, LHS, RHS and logic type transformation definition. This function is recommended to run recursively until the search pattern no longer exists.

5. Case Study

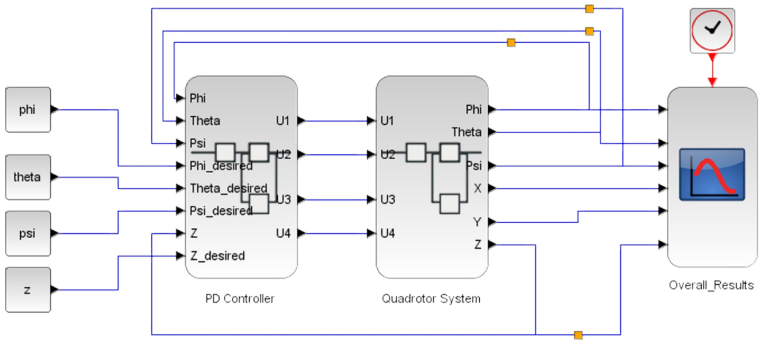


Fig. 6. Quadrotor Flight Dynamics and Control Model for the Case Study

In this section, the proposed approach will be exemplified with a sample case study that presents a typical nontrivial refactoring. The Scilab/Xcos model that will be presented in this case is for a Quadrotor Flight Dynamics and Control Model (Figure 6) which implements flight dynamics and control algorithms from Samirs work [48]. The model is composed of two super blocks, one for the control algorithms and the second one for the quadrotor flight dynamics. The inputs of the model are the desired attitude and the altitude of the quadrotor whereas the outputs are the actual attitude and the position. Two refactoring scenarios will be presented with this model.

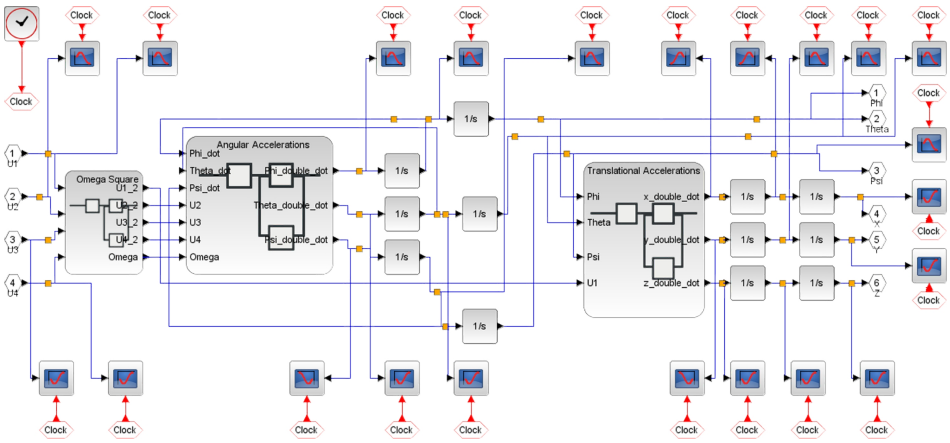


Fig. 7. Quadrotor Flight Dynamics Superblock

In the first refactoring scenario, the modeler would like to make a Monte Carlo simulation for the robustness analysis of the simulated quadrotor system. His model contains Scope blocks that he uses to observer the signals of interest (Figure 7). The first refactoring task is to replace all 22 *Scopes* with *To Workspace* blocks so that he can save the values from all signals of interest for all simulation executions. While the manual refactoring seems well applicable, modeler needs to repeat the refactoring task every time he makes a change in his model before he reruns his Monte Carlo simulation. The proposed refactoring approach enables him to define his refactoring task with a script and apply it as required in order to avoid labor intensive and error prone manual refactoring.

In the refactoring script, the modeler needs to identify all the *Scopes* blocks and replace them with *To Workspace*. The proposed set of functions enabled the user to conduct this refactoring task in more than one way. While it can be conducted just using the *find_and_replace* composite model transformation function, it is also possible for the modeler to write a script using the atomic model transformation

functions. A sample implementation is presented below:

```
//import the diagrams to be refactored
importXcosDiagram('quadrotor.xcos')
dl = get_diagrams(scs_m)
//no particular selection constraints
attribute_list=[]
constraint_list[]
//get a scope block
scope_block=CSCOPE("define")
//get a to workspace block
tows_block = TOWS_c("define")
//for every diagram
for i = 1: size("dl")
 //get the list of scope blocks
 scope_list=find_block(dl(i),scope_block,attribute_list,...
  constraint_list)
 //replace the scopeblocks with
 for j = 1: size("scope_list")
  replace_block(dl(i),scope_list(j),tows_block)
end; end
```

While the *replace_block* is a simple assignment operation, the implementation of *find_block* deserves a closer look. As presented in the listing below, each *block* in the *diagram* is checked against the search *block* using the block type information from the *gui* attribute. Then, if there is a selection constraint on an attribute, it is executed using the regular expression function of Scilab. The indexes of the matching *blocks* are listed in *block_list*.

```
function block_list=find_block(scs_m,block,attribute_list,...
 constraint_list)
//find block regarding the selection constraints
 block_list = []
 for i = 1:size(scs_m.objs) //for every object in the diagram
  if typeof(scs_m.objs(i))=="Block" then //when it is a block
   if scs_m.objs(i).gui==block.gui then //if block type matches
    if size(attribute_list,"r") > 0 then //check any constraints
     for j=1: size(attribute_list,"r")
      //get the constraining attribute
      [attr, dummy]=strsplit(attribute_list(j),'.',2)
      //get the field names
      if attr(1)=='model' then
       di =scs_m.objs(i).model
       field_names =fieldnames(scs_m.objs(i).model)
      else
```

```
        di =scs_m.objs(i).graphics
        field_names =fieldnames(scs_m.objs(i).graphics)
       end
       //find the index of the constraining attribute
       for k = 1:size(field_names,"r")
        if field_names(k)==attr(2)
         field_value=strcat(string(getfield((k), di)),',')
         //execute the regular expression
         if regexp(field_value, constraint_list(j))
          block_list=[block_list, i] // add the block if it matches
        end; end ; end; end
    else //no selection rules
     block_list=[block_list, i] //add the block to the list
 end; end; end; end;
endfunction
```

The second refactoring scenario task targets at optimizing the diagram against a particular bad smell. Sometimes, rather than using a gain block, which multiplies its input signal with the constant value defined as the parameter of the block, modelers use explicitly product blocks to multiply a signal with a constant value. While it is mathematically the same, gain blocks enhance the readability by reducing the number of blocks. These type of bad practices can be introduced to a model any time during its development and maintenance, constant review and refactoring is required to keep the model in shape. Manual review and refactoring take time and can not guarantee a full coverage. The proposed approach equips the modeler with the capability to define refactorings for bad smells of importance and apply them regularly execute.

Figure 8 depicts the *Omega Square* superblock from the model. Only in this superblock, there are eight instances of this bad practice. For this refactoring task, the modeler needs to develop a particular script that requires changing a parameter of the new block, namely the value of the gain, depending on the value of the removed constant block. In the sample implementation that is presented in the following listing, *find_subdiagram* is employed to identify the blocks that matches the specified pattern. The blocks and the associated links are first removed from the diagram. Then a gain block is added and required links are constructed. The script searches for a matching sub-diagram and conducts the replacement operation until no match is found.

```
//import the searched sub-diagram
importXcosDiagram('productwithconstantblock.xcos')
seached_subdiagram = scs_m
//import the diagrams to be refactored
importXcosDiagram('quadrotor.xcos')
dl = get_diagrams(scs_m)
```

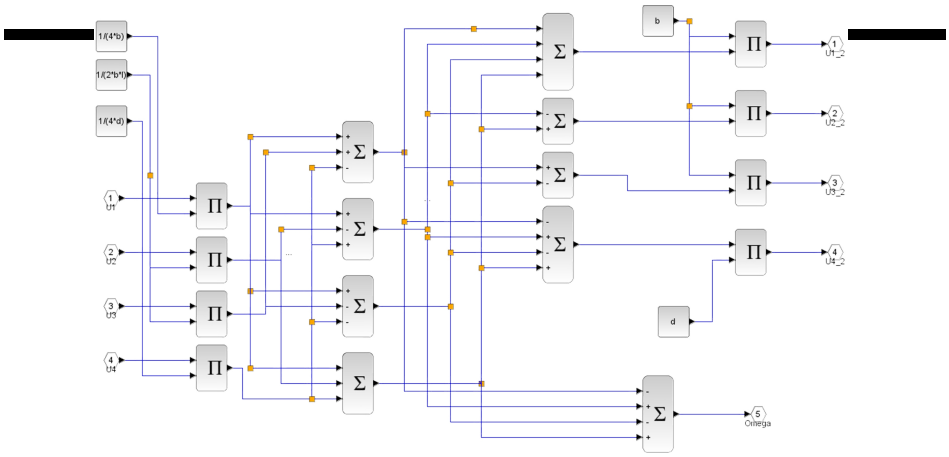


Fig. 8. Omega Square Superblock

```
//not particular selection constraints
attribute_list=[]
constraint_list[]
//for every diagram
for i = 1: size("dl")
 //get the list of matched blocks and links
 [matched_block_list matched_link_list]=find_subdiagram(dl(i),...
  searched_subdiagram,attribute_list,constraint_list)
 //get the index of const and product
 while size(matched_block_list)>0
  for j=1:size(matched_block_list)
   if dl(i).objs(matched_block_list(j)).gui=="CONST"
    const_index = j;
   elseif diagram_list(i).objs(matched_block_list(j)).gui=="PRODUCT"
    product_index = j;
  end; end
  //get the value for the gain
  gain_value = dl(i).objs(matched_block_list(const_index)).model.rpar
  //delete the const block
  delete_block(dl(i),matched_block_list(const_index))
  gain_block = GAINBLK("define") //create a gain block
  gain_block.model.rpar= gain_value //set the gain value
  //replace it with the product block
  replace(dl(i),matched_block_list(product_index),gain_block)
  //check for any other match
  [matched_block_list matched_link_list]=find_subdiagram(dl(i),...
```

```
    searched_subdiagram , attribute_list , constraint_list )
end ; end
```

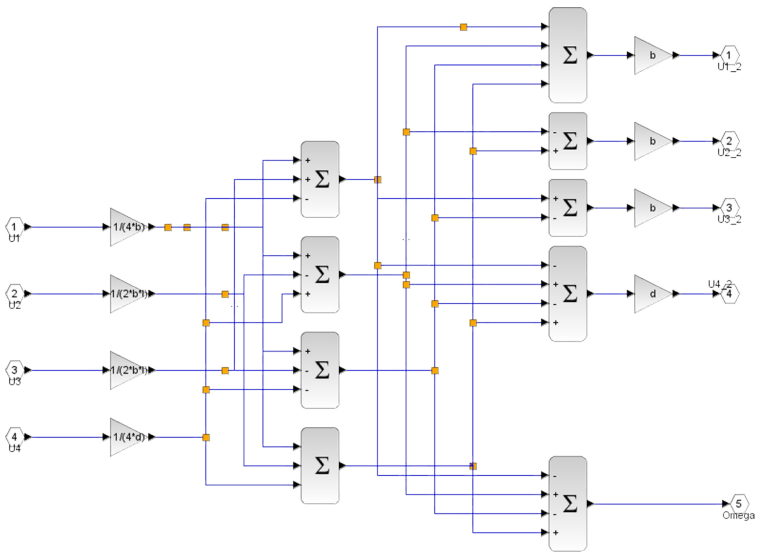


Fig. 9. Refactored Omega Square Superblock

This refactoring tasks produce a diagram as represented in Figure 9. These two refactoring tasks indicate that the proposed Scilab scripting API for pragmatic in-place model transformations is capable of providing modelers with a refactoring toolset. Modelers can utilize either composite model transformation functions or atomic ones in scripts to conduct refactoring operations in their models. This approach empowers modelers to compile a collection of refactoring scripts that they can apply to their models as required. The Scilab scripting API based approach also facilitates the adaptation of available refactoring scripts from the API level to overall refactoring task level. Therefore, this approach is regarded as more efficient that manual refactoring which is inherently not repeatable and adaptable. Besides, it can be claimed that find and replace capabilities provided over the API is more effective then manual reviewing and refactoring of the models, since it is always a possibility to fail while identifying the blocks or patterns in a complex and large model. Consider that the simple flight dynamics model that is used in this case study is composed of more than 600 features, namely blocks and links. Models used in industry can easily reach couple of thousand. Then the manual refactoring becomes much less efficient and much less effective.

6. Conclusion

This paper presents a refactoring approach for Scilab/Xcos models based on pragmatic in-place model transformations, which are carried out via atomic and composite model transformation functions, provided as a Scilab scripting level Application Programming Interface. This effort can be described as an early attempt that targets Scilab/Xcos. Further, it contributes to the related work on model refactoring that all target MATLAB/Simulink by concentrating on practical considerations as well as the theoretical background. Rather than employing well-developed model transformation tools and languages, it provides native Scilab model transformation functions, thus makes the transformation steps accessible and adaptable for the modeler.

Refactoring is regarded as a technical modernization activity. Aligned with the Architecture Driven Modernization approach of the Object Management Group, a model-based refactoring methodology is constructed based upon model transformations. First, a metamodeling hierarchy is defined for Scilab refactoring. Then the related metamodels are introduced. After presenting the specification and transformation approach based upon the metamodels, the concept has been implemented as a prototype and exercised with a case study. Thus, valuable evidence is collected about the success of the approach. While the atomic model transformation functions enable a modeler to tailor the refactoring operation according to particular refactoring requirements, composite model transformation functions provide ready to use high level functionality. Keeping all the transformation steps all in Scilab increases the accessibility of the refactoring operation and enables the adaptability of the refactoring scripts.

On the other hand, the implementation that is presented in this paper basically targets a concept demonstration and evaluation. Therefore, it can be called a prototype. A more robust implementation is required to offer the approach and the toolset for Scilab/Xcos user community. Further effort is required to make the implementation more efficient and less error prone. In this respect, graph pattern matching techniques require further attention. The toolset also needs to be supported by an automatic layouting algorithm, thus, after any refactoring operation the graphical representation is beautified.

Future work aims at enhancing the proposed toolset for typical refactoring operations. This includes automatic layouting, clone detection and, guideline checking and repair. It is also further planned to apply the approach to large industrial Scilab/Xcos models and enhance the implementation with end user feedback. Thus the toolset is intended to be matured in order to be introduced as an integrated part of Scilab distribution. Finally, to establish a wider refactoring practice in model engineering, the approach can be be adapted to other simulation modeling tools environments.

References

1. Stürmer I., Conrad, M., Fey, I., Dörr, H., Experiences with Model and Autocode Reviews in Model-based Software Development, *Proceedings of the 2006 International Workshop on Software Engineering for Automotive Systems*, Shanghai, China, pp. 45–52, 2006.
2. Simulink User's Guide `http://www.mathworks.de/help/pdf_doc/simulink/sl_using.pdf`.
3. Xcos for very beginners `https://www.scilab.org/content/download/1107/10095/file/Xcos_beginners.pdf`.
4. Tran Q. M., Wilmes B., Dziobek C., Refactoring of Simulink diagrams via composition of transformation steps, *ICSEA 2013, The Eighth International Conference on Software Engineering Advances*, Venice, Italy, pp. 140–145, 2013.
5. Zhang L., Shen Y.W., Zhang X.S., Song X, Tao F., The model engineering for complex system simulation, *The 26th European Modeling & Simulation Symposium*, Bordeaux, France, 2014.
6. Zeigler B.P., Zhang L., Service-Oriented Model Engineering and Simulation for System of Systems Engineering, Eds. Yilmaz L., Concepts and Methodologies for Modeling and Simulation, Springer International Publishing, pp. 19–44, 2015.
7. Gestwa M., Höfinger, M., Lantzsch, R., Alvermann, K., The Software Development Environment of the DLR Research Rotorcraft ACT/FHS, *Deutscher Luft- und Raumfahrtkongress 2012*, Berlin, Germany, 2012.
8. Gerlach T., Durak, U., Gotschlich, J., Model Integration Workflow for Keeping Models upto Date in a Research Simulator, *Proceedings of the 4th International Conference on Simulation and Modeling Methodologies, Technologies and Applications*, Vienna, Austria, pp. 125–132, 2014.
9. Chikofsky E. J., Cross J. H., Reverse engineering and design recovery: A taxonomy, *IEEE Software* **7(1)**:13–17, 1990.
10. Fowler M. *Refactoring: Improving the Design of Existing Code*, Addison Wesley , Reading, 1999.
11. Bergey J, Smith D., Weiderman N., Woods S., Options analysis for reengineering (OAR): Issues and conceptual approach, *DTIC Document*, 1999.
12. Pérez-Castillo R. and de Guzmán I. G. R., Piattini M., Architecture-Driven Modernization, Eds. Dogru A.H. *et al.*, Modern Software Engineering Concepts and Practices: Advanced Approaches, IGI Global, 2010.
13. Khusidman V., Adm transformation, *Object Management Group Technical report*, 2008.
14. Mens T., Tourwé T., A survey of software refactoring, *IEEE Transactions on Software Engineering* **30(2)**:126–139, 2004.
15. Denil J., Mosterman P. J., Vangheluwe, H., Rule-based model transformation for, and in simulink, *Proceedings of the Symposium on Theory of Modeling & Simulation-DEVS Integrative*, San Diego, CA, pp. 314–321, 2014.
16. Amelunxen C., Legros E., Schürr A., Stürmer I., Checking and enforcement of modeling guidelines with graph transformations, Applications of Graph Transformations with Industrial Relevance, Springer, pp. 313-328, 2008.
17. Stürmer I., Kreuz I., Schäfer W., Schürr A., The MATE Approach: Enhanced Simulink® and Stateflow® Model Transformation, *Proceedings of MathWorks Automotive Conference*, Dearborn, MI, 2007.
18. Stürmer I., Travkin D., Automated Transformation of MATLAB Simulink and Stateflow Models, *Proc. of 4th Workshop on Object-oriented Modeling of Embedded Real-time Systems*, Paderborn, Germany, pp. 57–62, 2007.

19. Farkas T., Hein C., Ritter T., Automatic Evaluation of Modelling Rules and Design Guidelines, *Proc. of the Workshop From code centric to model centric software engineering: Practices, Implications and ROI*, Bilbago, Spain, 2006.
20. Giese H., Meyer M., Wagner R., A prototype for guideline checking and model transformation in Matlab/Simulink, *The 4th International Fujaba Days*, Bayreuth, Germany, pp. 56–60, 2006.
21. Sarjoughian H.S., Elamvazhuthi V., CoSMoS: a visual environment for component-based modeling, experimental design, and simulation, *Proceedings of the 2nd International Conference on Simulation Tools and Techniques (Simutools'09)*, Rome, Italy, 2009.
22. Seo C., Zeigler B.P., Coop R., Kim D., DEVS modeling and simulation methodology with MS4 Me software tool, *Proceedings of the Symposium on Theory of Modeling & Simulation - DEVS Integrative M & S Symposium (TMS'13)*, San Diego, CA, 2009.
23. Bergero F., Kofman E., PowerDEVS: a tool for hybrid system modeling and real-time simulation, *Simulation* **87(1-2)**:113–132, 2011.
24. Fritzson P., Aronsson P., Pop A., Lundvall H., Nystrom K., Saldamli L., Broman D., Sandholm A., OpenModelica-A free open-source environment for system modeling, simulation, and teaching, *Proceedings of the IEEE International Symposium on Computer-Aided Control Systems Design*, Munich, Germany, 2006.
25. Akesson J.,Gafvert M., Tummescheit H., OpenModelica-A free open-source environment for system modeling, simulation, and teaching, *Proceedings of the 6th Vienna International Conference on Mathematical Modelling*, Vienna, Austria, 2009.
26. Scilab Online Help `https://help.scilab.org/docs/5.5.1/en_US/index.html`.
27. Roy C. K., Cordy J. R., Koschke R., Comparison and Evaluation of Code Clone Detection Techniques and Tools: A Qualitative Approach, *Science of Computer Programming* **74(7)**:470–498, 2009.
28. Deissenboeck F., Hummel B., Juergens E., Schaetz B., Wagner S., Girard J. F., Teuchert S., Clone detection in automotive model-based development, *The 30th International Conference on Software Engineering(ICSE 2008)*, Leipzig, Germany, 2008.
29. Deissenboeck F., Hummel B., Juergens E., Pfaehler M., Schaetz B., Model clone detection in practice, *The 4th International Workshop on Software Clones*, Cape Town, South Africa, 2010.
30. Alalfi M. H., Cordy J. R., Dean T. R., Stephan, M., Stevenson A., Models are code too: Near-miss clone detection for Simulink models, *The 28th IEEE International Conference on Software Maintenance*, Trento, Italy, 2012.
31. Stephan M., Alafi M. H., Stevenson A., Cordy, J. R. , Towards qualitative comparison of simulink model clone detection approaches, *The 6th International Workshop on Software Clones*, Zurich, Switzerland, 2012.
32. Schmidt A., Durak U., Rasch C., Pawletta T., Model-Based Testing Approach for MATLAB/Simulink using System Entity Structure and Experimental Frames, *The 5th International Workshop on Model-driven Approaches for Simulation Engineering (Mod4Sim'15)*, Alexandria, VA, 2015.
33. Zander-Nowicka J. *Model-based Testing of Real-Time Embedded Systems in the Automotive Domain*, Fraunhofer IRB Verlag , Berlin, Germany, 2009.
34. Porres I. *Model Refactorings as Rule-Based Update Transformations*, Turku Center for Computer Science , Turku, Finland, 2003.
35. Zhang J., Lin Y., Gray, J., Generic and Domain-Specific Model Refactoring using a Model Transformation Engine, Eds. Beydeda S., Book M., Gruhn V., Model-Driven Software Development, Springer, 2005.
36. Sunye G., Pollet D., Le Traon Y., Jezequel J. M.,Refactoring UML models, *The 4th*

International Conference on the Unified Modeling Language(¡¡UML¿¿2001), Toronto, Canada, 2001.
37. Nickel U., Niere J., Zndorf A., Tool demonstration: The FUJABA environment, *Proc. of the 22nd International Conference on Software Engineering (ICSE)*, Limerick, Ireland, 2000.
38. Simulink Verification and Validation User's Guide `http://de.mathworks.com/help/pdf_doc/slvnv/slvnv_ug.pdf`.
39. MathWorks Automotive AdvisoryBoard, Control Algorithm Modeling Guidelines Using MATLAB®, Simulink®, and Stateflow® V3.1, *The Mathworks, Inc*, 2015.
40. Czarnecki K., Helsen S., Feature-based survey of model transformation approaches, *IBM Systems Journal*, **45(3)**:621–645, 2006.
41. Campbell S. L., Chancelier J. P., Nikoukhah R. *Modeling and Simulation in Scilab/Scicos*, Springer, New York, NY, 2006.
42. Kühne T., Mezei G., Syriani E., Vangheluwe H., Wimmer M., Explicit Transformation Modeling, Eds. Ghosh S., Models in Software Engineering, Springer Berlin Heidelberg,2010.
43. Eclipse Modeling Framework (EMF) `https://www.eclipse.org/modeling/emf/`.
44. Steinber, D., Budinsky, F., Paternostro M. ,Merks, E. *EMF: Eclipse Modeling Framework*, Addison-Wesley , Upper Saddle River, NJ, 2006.
45. Czarnecki K.., Helsen S., Classification of Model Trasformation Approaches, *Proceedings of OOPSLA'03 Workshop on Generative Techniques in Context of Model-Driven Architecture*, Anaheim, CA, 2003.
46. Watt A. *Beginning Regular Expressions*, Wiley Publishing, Inc., Indianapolis, IN, 2005.
47. Bauldin M., Programming in Scilab, *Scilab Consortium*, 2010.
48. Samir B., Design and Control of Quadrotors with Application to Autonomous Flying, *Ph.D. Thesis. Lausanne: cole Polytechnique Fdrale de Lausanne*, 2007.